

## MSD Script

Generated by Doxygen 1.10.0



# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Catch::Detail::Approx	??
Catch::Generators::as< T >	??
Catch::AssertionHandler	??
Catch::AssertionInfo	??
Catch::AssertionReaction	??
Catch::Capturer	??
Catch::Matchers::StdString::CasedString	??
Catch::CaseSensitive	??
Catch_global_namespace_dummy	??
Catch::Counts	??
Catch::Decomposer	??
Catch::Detail::EnumInfo	??
std::exception	
Catch::GeneratorException	??
Catch::ExceptionTranslatorRegistrar	??
Expr	??
Add	??
Mult	??
Num	??
Var	??
Catch::ExprLhs< LhsT >	??
std::false_type	
Catch::always_false< T >	??
Catch::detail::is_range_impl< T, typename >	??
Catch::is_range< T >	??
Catch::Generators::GeneratorUntypedBase	??
Catch::Generators::IGenerator< std::vector< T > >	??
Catch::Generators::ChunkGenerator< T >	??
Catch::Generators::IGenerator< Float >	??
Catch::Generators::RandomFloatingGenerator< Float >	??
Catch::Generators::IGenerator< Integer >	??
Catch::Generators::RandomIntegerGenerator< Integer >	??
Catch::Generators::IGenerator< T >	??
Catch::Generators::FilterGenerator< T, Predicate >	??

Catch::Generators::FixedValuesGenerator< T > . . . . .	??
Catch::Generators::Generators< T > . . . . .	??
Catch::Generators::IteratorGenerator< T > . . . . .	??
Catch::Generators::MapGenerator< T, U, Func > . . . . .	??
Catch::Generators::RangeGenerator< T > . . . . .	??
Catch::Generators::RepeatGenerator< T > . . . . .	??
Catch::Generators::SingleValueGenerator< T > . . . . .	??
Catch::Generators::TakeGenerator< T > . . . . .	??
Catch::Generators::GeneratorWrapper< T > . . . . .	??
Catch::Generators::GeneratorWrapper< U > . . . . .	??
Catch::IContext . . . . .	??
Catch::IMutableContext . . . . .	??
Catch::IExceptionTranslator . . . . .	??
Catch::IExceptionTranslatorRegistry . . . . .	??
Catch::IGeneratorTracker . . . . .	??
Catch::IMutableEnumValuesRegistry . . . . .	??
Catch::IMutableRegistryHub . . . . .	??
Catch::IRegistryHub . . . . .	??
Catch::IResultCapture . . . . .	??
Catch::IRunner . . . . .	??
Catch::is_callable< T > . . . . .	??
Catch::is_callable< Fun(Args...)> . . . . .	??
Catch::is_callable_tester . . . . .	??
Catch::Detail::IsStreamInsertable< T > . . . . .	??
Catch::IStream . . . . .	??
Catch::ITestCaseRegistry . . . . .	??
Catch::ITestInvoker . . . . .	??
Catch::TestInvokerAsMethod< C > . . . . .	??
Catch::ITransientExpression . . . . .	??
Catch::BinaryExpr< LhsT, RhsT > . . . . .	??
Catch::MatchExpr< ArgT, MatcherT > . . . . .	??
Catch::UnaryExpr< LhsT > . . . . .	??
Catch::LazyExpression . . . . .	??
Catch::Matchers::Impl::MatcherMethod< ObjectT > . . . . .	??
Catch::Matchers::Impl::MatcherBase< std::exception > . . . . .	??
Catch::Matchers::Impl::MatcherBase< double > . . . . .	??
Catch::Matchers::Impl::MatcherBase< ArgT > . . . . .	??
Catch::Matchers::Impl::MatchAllOf< ArgT > . . . . .	??
Catch::Matchers::Impl::MatchAnyOf< ArgT > . . . . .	??
Catch::Matchers::Impl::MatchNotOf< ArgT > . . . . .	??
Catch::Matchers::Impl::MatcherBase< std::string > . . . . .	??
Catch::Matchers::Impl::MatcherMethod< ArgT > . . . . .	??
Catch::Matchers::Impl::MatcherMethod< double > . . . . .	??
Catch::Matchers::Impl::MatcherMethod< std::exception > . . . . .	??
Catch::Matchers::Impl::MatcherMethod< std::string > . . . . .	??
Catch::Matchers::Impl::MatcherMethod< T > . . . . .	??
Catch::Matchers::Impl::MatcherBase< std::vector< T, AllocMatch > > . . . . .	??
Catch::Matchers::Impl::MatcherBase< std::vector< T, Alloc > > . . . . .	??
Catch::Matchers::Impl::MatcherBase< T > . . . . .	??
Catch::Matchers::Exception::ExceptionMessageMatcher . . . . .	??
Catch::Matchers::Floating::WithinAbsMatcher . . . . .	??
Catch::Matchers::Floating::WithinRelMatcher . . . . .	??
Catch::Matchers::Floating::WithinUlpsMatcher . . . . .	??
Catch::Matchers::Generic::PredicateMatcher< T > . . . . .	??
Catch::Matchers::StdString::RegexMatcher . . . . .	??
Catch::Matchers::StdString::StringMatcherBase . . . . .	??
Catch::Matchers::StdString::ContainsMatcher . . . . .	??

Catch::Matchers::StdString::EndsWithMatcher	??
Catch::Matchers::StdString::EqualsMatcher	??
Catch::Matchers::StdString::StartsWithMatcher	??
Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch >	??
Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc >	??
Catch::Matchers::Vector::ContainsMatcher< T, AllocComp, AllocMatch >	??
Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch >	??
Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch >	??
Catch::Matchers::Impl::MatcherUntypedBase	??
Catch::Matchers::Impl::MatcherBase< std::exception >	??
Catch::Matchers::Impl::MatcherBase< double >	??
Catch::Matchers::Impl::MatcherBase< ArgT >	??
Catch::Matchers::Impl::MatcherBase< std::string >	??
Catch::Matchers::Impl::MatcherBase< std::vector< T, AllocMatch > >	??
Catch::Matchers::Impl::MatcherBase< std::vector< T, Alloc > >	??
Catch::Matchers::Impl::MatcherBase< T >	??
Catch::MessageInfo	??
Catch::MessageStream	??
Catch::MessageBuilder	??
Catch::NameAndTags	??
Catch::NonCopyable	??
Catch::AutoReg	??
Catch::IConfig	??
Catch::ReusableStringStream	??
Catch::Section	??
Catch::Option< T >	??
Catch::pluralise	??
Catch::RegistrarForTagAliases	??
Catch::ResultDisposition	??
Catch::ResultWas	??
Catch::RunTests	??
Catch::ScopedMessage	??
Catch::SectionEndInfo	??
Catch::SectionInfo	??
Catch::ShowDurations	??
Catch::SimplePcg32	??
Catch::SourceLineInfo	??
Catch::StreamEndStop	??
Catch::StringMaker< T, typename >	??
Catch::StringMaker< bool >	??
Catch::StringMaker< Catch::Detail::Approx >	??
Catch::StringMaker< char * >	??
Catch::StringMaker< char >	??
Catch::StringMaker< char const * >	??
Catch::StringMaker< char[SZ]>	??
Catch::StringMaker< double >	??
Catch::StringMaker< float >	??
Catch::StringMaker< int >	??
Catch::StringMaker< long >	??
Catch::StringMaker< long long >	??
Catch::StringMaker< R C::* >	??
Catch::StringMaker< R, typename std::enable_if< is_range< R >::value &&!::Catch::Detail::IsStream← Insertable< R >::value >::type >	??
Catch::StringMaker< signed char >	??
Catch::StringMaker< signed char[SZ]>	??
Catch::StringMaker< std::nullptr_t >	??
Catch::StringMaker< std::string >	??

Catch::StringMaker< std::wstring > . . . . .	??
Catch::StringMaker< T * > . . . . .	??
Catch::StringMaker< T[SZ]> . . . . .	??
Catch::StringMaker< unsigned char > . . . . .	??
Catch::StringMaker< unsigned char[SZ]> . . . . .	??
Catch::StringMaker< unsigned int > . . . . .	??
Catch::StringMaker< unsigned long > . . . . .	??
Catch::StringMaker< unsigned long long > . . . . .	??
Catch::StringMaker< wchar_t * > . . . . .	??
Catch::StringMaker< wchar_t const * > . . . . .	??
Catch::StringRef . . . . .	??
Catch::TestCaseInfo . . . . .	??
Catch::TestCase . . . . .	??
Catch::TestFailureException . . . . .	??
Catch::Timer . . . . .	??
Catch::Totals . . . . .	??
std::true_type	
Catch::detail::is_range_impl< T, typename void_type< decltype(begin(std::declval< T >()))>::type > . . . . .	??
Catch::true_given< typename > . . . . .	??
Catch::UseColour . . . . .	??
Catch::detail::void_type<... > . . . . .	??
Catch::WaitForKeypress . . . . .	??
Catch::WarnAbout . . . . .	??

## Chapter 2

# Class Index

### 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">Add</a>	??
<a href="#">Catch::always_false&lt; T &gt;</a>	??
<a href="#">Catch::Detail::Approx</a>	??
<a href="#">Catch::Matchers::Vector::ApproxMatcher&lt; T, AllocComp, AllocMatch &gt;</a>	??
<a href="#">Catch::Generators::as&lt; T &gt;</a>	??
<a href="#">Catch::AssertionHandler</a>	??
<a href="#">Catch::AssertionInfo</a>	??
<a href="#">Catch::AssertionReaction</a>	??
<a href="#">Catch::AutoReg</a>	??
<a href="#">Catch::BinaryExpr&lt; LhsT, RhstT &gt;</a>	??
<a href="#">Catch::Capturer</a>	??
<a href="#">Catch::Matchers::StdString::CasedString</a>	??
<a href="#">Catch::CaseSensitive</a>	??
<a href="#">Catch_global_namespace_dummy</a>	??
<a href="#">Catch::Generators::ChunkGenerator&lt; T &gt;</a>	??
<a href="#">Catch::Matchers::Vector::ContainsElementMatcher&lt; T, Alloc &gt;</a>	??
<a href="#">Catch::Matchers::StdString::ContainsMatcher</a>	??
<a href="#">Catch::Matchers::Vector::ContainsMatcher&lt; T, AllocComp, AllocMatch &gt;</a>	??
<a href="#">Catch::Counts</a>	??
<a href="#">Catch::Decomposer</a>	??
<a href="#">Catch::Matchers::StdString::EndsWithMatcher</a>	??
<a href="#">Catch::Detail::EnumInfo</a>	??
<a href="#">Catch::Matchers::StdString::EqualsMatcher</a>	??
<a href="#">Catch::Matchers::Vector::EqualsMatcher&lt; T, AllocComp, AllocMatch &gt;</a>	??
<a href="#">Catch::Matchers::Exception::ExceptionMessageMatcher</a>	??
<a href="#">Catch::ExceptionTranslatorRegistrar</a>	??
<a href="#">Expr</a>	??
<a href="#">Catch::ExprLhs&lt; LhsT &gt;</a>	??
<a href="#">Catch::Generators::FilterGenerator&lt; T, Predicate &gt;</a>	??
<a href="#">Catch::Generators::FixedValuesGenerator&lt; T &gt;</a>	??
<a href="#">Catch::GeneratorException</a>	??
<a href="#">Catch::Generators::Generators&lt; T &gt;</a>	??
<a href="#">Catch::Generators::GeneratorUntypedBase</a>	??
<a href="#">Catch::Generators::GeneratorWrapper&lt; T &gt;</a>	??
<a href="#">Catch::IConfig</a>	??

Catch::IContext	??
Catch::IExceptionTranslator	??
Catch::IExceptionTranslatorRegistry	??
Catch::Generators::IGenerator< T >	??
Catch::IGeneratorTracker	??
Catch::IMutableContext	??
Catch::IMutableEnumValuesRegistry	??
Catch::IMutableRegistryHub	??
Catch::IRegistryHub	??
Catch::IResultCapture	??
Catch::IRunner	??
Catch::is_callable< T >	??
Catch::is_callable< Fun(Args...)>	??
Catch::is_callable_tester	??
Catch::is_range< T >	??
Catch::detail::is_range_impl< T, typename >	??
Catch::detail::is_range_impl< T, typename void_type< decltype(begin(std::declval< T >()))>::type >	??
Catch::Detail::IsStreamInsertable< T >	??
Catch::IStream	??
Catch::Generators::IteratorGenerator< T >	??
Catch::ITestCaseRegistry	??
Catch::ITestInvoker	??
Catch::ITransientExpression	??
Catch::LazyExpression	??
Catch::Generators::MapGenerator< T, U, Func >	??
Catch::Matchers::Impl::MatchAllOf< ArgT >	??
Catch::Matchers::Impl::MatchAnyOf< ArgT >	??
Catch::Matchers::Impl::MatcherBase< T >	??
Catch::Matchers::Impl::MatcherMethod< ObjectT >	??
Catch::Matchers::Impl::MatcherUntypedBase	??
Catch::MatchExpr< ArgT, MatcherT >	??
Catch::Matchers::Impl::MatchNotOf< ArgT >	??
Catch::MessageBuilder	??
Catch::MessageInfo	??
Catch::MessageStream	??
Mult	??
Catch::NameAndTags	??
Catch::NonCopyable	??
Num	??
Catch::Option< T >	??
Catch::pluralise	??
Catch::Matchers::Generic::PredicateMatcher< T >	??
Catch::Generators::RandomFloatingGenerator< Float >	??
Catch::Generators::RandomIntegerGenerator< Integer >	??
Catch::Generators::RangeGenerator< T >	??
Catch::Matchers::StdString::RegexMatcher	??
Catch::RegistrarForTagAliases	??
Catch::Generators::RepeatGenerator< T >	??
Catch::ResultDisposition	??
Catch::ResultWas	??
Catch::ReusableStringStream	??
Catch::RunTests	??
Catch::ScopedMessage	??
Catch::Section	??
Catch::SectionEndInfo	??
Catch::SectionInfo	??
Catch::ShowDurations	??
Catch::SimplePcg32	??



Catch::Generators::SingleValueGenerator< T >	??
Catch::SourceLineInfo	??
Catch::Matchers::StdString::StartsWithMatcher	??
Catch::StreamEndStop	??
Catch::StringMaker< T, typename >	??
Catch::StringMaker< bool >	??
Catch::StringMaker< Catch::Detail::Approx >	??
Catch::StringMaker< char * >	??
Catch::StringMaker< char >	??
Catch::StringMaker< char const * >	??
Catch::StringMaker< char[SZ]>	??
Catch::StringMaker< double >	??
Catch::StringMaker< float >	??
Catch::StringMaker< int >	??
Catch::StringMaker< long >	??
Catch::StringMaker< long long >	??
Catch::StringMaker< R C::* >	??
Catch::StringMaker< R, typename std::enable_if< is_range< R >::value &&!::Catch::Detail::IsStreamInsertable< R >::value >::value >	??
Catch::StringMaker< signed char >	??
Catch::StringMaker< signed char[SZ]>	??
Catch::StringMaker< std::nullptr_t >	??
Catch::StringMaker< std::string >	??
Catch::StringMaker< std::wstring >	??
Catch::StringMaker< T * >	??
Catch::StringMaker< T[SZ]>	??
Catch::StringMaker< unsigned char >	??
Catch::StringMaker< unsigned char[SZ]>	??
Catch::StringMaker< unsigned int >	??
Catch::StringMaker< unsigned long >	??
Catch::StringMaker< unsigned long long >	??
Catch::StringMaker< wchar_t * >	??
Catch::StringMaker< wchar_t const * >	??
Catch::Matchers::StdString::StringMatcherBase	??
Catch::StringRef	??
Catch::Generators::TakeGenerator< T >	??
Catch::TestCase	??
Catch::TestCaseInfo	??
Catch::TestFailureException	??
Catch::TestInvokerAsMethod< C >	??
Catch::Timer	??
Catch::Totals	??
Catch::true_given< typename >	??
Catch::UnaryExpr< LhsT >	??
Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch >	??
Catch::UseColour	??
Var	??
Catch::detail::void_type<... >	??
Catch::WaitForKeypress	??
Catch::WarnAbout	??
Catch::Matchers::Floating::WithinAbsMatcher	??
Catch::Matchers::Floating::WithinRelMatcher	??
Catch::Matchers::Floating::WithinUlpMatcher	??



# Chapter 3

## File Index

### 3.1 File List

Here is a list of all documented files with brief descriptions:

/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h . . . . .	??
/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/cmdline.h . . . . .	??
/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.cpp This header file declares the Expr class hierarchy for representing arithmetic expressions. It includes the abstract base class Expr and its derived classes Num, Var, Add, and Mult. Each class implements functionalities for equality comparison, interpretation (evaluation), variable presence checking, substitution, pretty printing, and standard printing of expressions. This architecture allows for the representation and manipulation of complex arithmetic expressions involving numbers, variables, and the operations of addition and multiplication . . . . .	??
/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.h This header file defines a hierarchy of expression classes for representing and manipulating arithmetic expressions . . . . .	??
/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/ExprTests.cpp This test file contains a series of test cases for the Expr class hierarchy, focusing on Var, Num, Add, and Mult classes . . . . .	??
/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/ExprTests.h . . . . .	??
/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/main.cpp Main entry point for the program. This file contains the main function that serves as the entry point of the program . . . . .	??

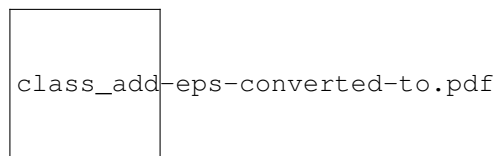


## Chapter 4

# Class Documentation

### 4.1 Add Class Reference

Inheritance diagram for Add:



#### Public Member Functions

- [Add](#) ([Expr](#) \*lhs, [Expr](#) \*rhs)  
*Constructor for the [Add](#) class. Creates the [Add](#) object with left and right expressions.*
- bool [equals](#) ([Expr](#) \*e)  
*Implementation of the equals function for [Add](#).*
- int [interp](#) ()  
*the [interp\(\)](#) function for [Add](#) class.*
- bool [has\\_variable](#) ()  
*Checks if the expression contains any variables.*
- [Expr](#) \* [subst](#) (string varName, [Expr](#) \*replacement)  
*Substitutes a variable within the expression with another expression.*
- virtual void [print](#) (ostream &os)  
*Prints the [Add](#) expression to an output stream.*
- void [pretty\\_print\\_at](#) (ostream &ot, precedence\_t prec)  
*Pretty prints the [Add](#) expression with correct precedence handling.*

#### Public Member Functions inherited from [Expr](#)

- string [to\\_string](#) ()
- void [pretty\\_print](#) (ostream &ostream)
- string [to\\_pretty\\_string](#) ()

## Public Attributes

- [Expr](#) \* lhs
- [Expr](#) \* rhs

## 4.1.1 Constructor & Destructor Documentation

### 4.1.1.1 Add()

```
Add::Add (
    Expr * lhs,
    Expr * rhs )
```

Constructor for the [Add](#) class. Creates the [Add](#) object with left and right expressions.

#### Parameters

<i>lhs</i>	The left expression.
<i>rhs</i>	The right expression.

## 4.1.2 Member Function Documentation

### 4.1.2.1 equals()

```
bool Add::equals (
    Expr * e ) [virtual]
```

Implementation of the equals function for [Add](#).

#### Parameters

<i>e</i>	the expression you compare.
----------	-----------------------------

#### Returns

false if add is a null pointer, true otherwise. Verifies the current [Var](#) object is equal to a different expression.

Implements [Expr](#).

### 4.1.2.2 has\_variable()

```
bool Add::has_variable ( ) [virtual]
```

Checks if the expression contains any variables.

#### Returns

True if either lhs or rhs contains a variable, false otherwise.

Implements [Expr](#).

#### 4.1.2.3 interp()

```
int Add::interp ( ) [virtual]
```

the [interp\(\)](#) function for [Add](#) class.

##### Returns

lefthand side and righthand side with the `Interp()` method.

Implements [Expr](#).

#### 4.1.2.4 pretty\_print\_at()

```
void Add::pretty_print_at (
    ostream & ot,
    precedence_t prec ) [virtual]
```

Pretty prints the [Add](#) expression with correct precedence handling.

##### Parameters

<i>o</i>	The output stream to print to.
<i>prec</i>	The precedence level of the expression's context.

Reimplemented from [Expr](#).

#### 4.1.2.5 print()

```
void Add::print (
    ostream & ostream ) [virtual]
```

Prints the [Add](#) expression to an output stream.

##### Parameters

<i>ostream</i>	The output stream to print to.
----------------	--------------------------------

Implements [Expr](#).

#### 4.1.2.6 subst()

```
Expr * Add::subst (
    string varName,
    Expr * replacement ) [virtual]
```

Substitutes a variable within the expression with another expression.

## Parameters

<i>varName</i>	The name of the variable to be substituted.
<i>replacement</i>	The expression to substitute in place of the variable.

## Returns

A new [Add](#) expression with the variable substituted.

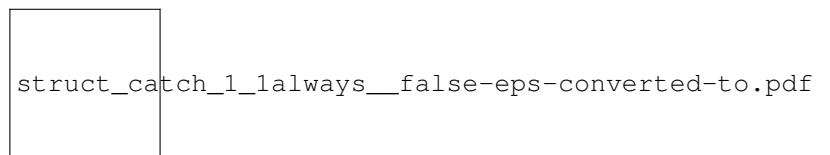
Implements [Expr](#).

The documentation for this class was generated from the following files:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.h](#)
- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.cpp](#)

## 4.2 `Catch::always_false< T >` Struct Template Reference

Inheritance diagram for `Catch::always_false< T >`:



The documentation for this struct was generated from the following file:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h](#)

## 4.3 `Catch::Detail::Approx` Class Reference

## Public Member Functions

- **Approx** ([double](#) value)
- **Approx operator-** ([\(\)](#) [const](#))
- `template<typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>`  
**Approx operator()** ([T const](#) &value) [const](#)
- `template<typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>`  
**Approx** ([T const](#) &value)
- `template<typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>`  
**Approx & epsilon** ([T const](#) &[newEpsilon](#))
- `template<typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>`  
**Approx & margin** ([T const](#) &[newMargin](#))
- `template<typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>`  
**Approx & scale** ([T const](#) &[newScale](#))
- `std::string toString` ([\(\)](#) [const](#))



## Static Public Member Functions

- [static Approx custom \(\)](#)

## Friends

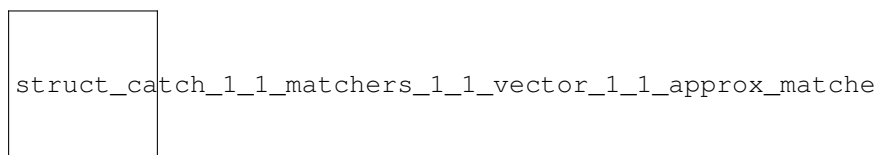
- [template<typename T , typename = typename std::enable\\_if<std::is\\_constructible<double, T>::value>::type> bool operator== \(const T &lhs, Approx const &rhs\)](#)
- [template<typename T , typename = typename std::enable\\_if<std::is\\_constructible<double, T>::value>::type> bool operator== \(Approx const &lhs, const T &rhs\)](#)
- [template<typename T , typename = typename std::enable\\_if<std::is\\_constructible<double, T>::value>::type> bool operator!= \(T const &lhs, Approx const &rhs\)](#)
- [template<typename T , typename = typename std::enable\\_if<std::is\\_constructible<double, T>::value>::type> bool operator!= \(Approx const &lhs, T const &rhs\)](#)
- [template<typename T , typename = typename std::enable\\_if<std::is\\_constructible<double, T>::value>::type> bool operator<= \(T const &lhs, Approx const &rhs\)](#)
- [template<typename T , typename = typename std::enable\\_if<std::is\\_constructible<double, T>::value>::type> bool operator<= \(Approx const &lhs, T const &rhs\)](#)
- [template<typename T , typename = typename std::enable\\_if<std::is\\_constructible<double, T>::value>::type> bool operator>= \(T const &lhs, Approx const &rhs\)](#)
- [template<typename T , typename = typename std::enable\\_if<std::is\\_constructible<double, T>::value>::type> bool operator>= \(Approx const &lhs, T const &rhs\)](#)

The documentation for this class was generated from the following file:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h](#)

## 4.4 Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch > Struct Template Reference

Inheritance diagram for Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch >:



## Public Member Functions

- [ApproxMatcher \(std::vector< T, AllocComp > const &comparator\)](#)
- [bool match \(std::vector< T, AllocMatch > const &v\) const override](#)
- [std::string describe \(\) const override](#)
- [template<typename = typename std::enable\\_if<std::is\\_constructible<double, T>::value>::type> ApproxMatcher & epsilon \(T const &newEpsilon\)](#)
- [template<typename = typename std::enable\\_if<std::is\\_constructible<double, T>::value>::type> ApproxMatcher & margin \(T const &newMargin\)](#)
- [template<typename = typename std::enable\\_if<std::is\\_constructible<double, T>::value>::type> ApproxMatcher & scale \(T const &newScale\)](#)

## Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)

- [MatchAllOf< T > operator&& \(MatcherBase const &other\) const](#)
- [MatchAnyOf< T > operator|| \(MatcherBase const &other\) const](#)
- [MatchNotOf< T > operator! \(\) const](#)

## Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- [MatcherUntypedBase \(MatcherUntypedBase const &\)=default](#)
- [MatcherUntypedBase & operator= \(MatcherUntypedBase const &\)=delete](#)
- [std::string toString \(\) const](#)

## Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- [virtual bool match \(T const &arg\) const=0](#)

## Public Attributes

- [std::vector< T, AllocComp > const & m\\_comparator](#)
- [Catch::Detail::Approx approx](#) = [Catch::Detail::Approx::custom\(\)](#)

## Additional Inherited Members

## Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- [std::string m\\_cachedToString](#)

## 4.4.1 Member Function Documentation

### 4.4.1.1 describe()

```
template<typename T , typename AllocComp , typename AllocMatch >
std::string Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch >::describe ( )
const [inline], [override], [virtual]
```

Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

The documentation for this struct was generated from the following file:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h](#)

## 4.5 [Catch::Generators::as< T >](#) Struct Template Reference

The documentation for this struct was generated from the following file:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h](#)

## 4.6 Catch::AssertionHandler Class Reference

### Public Member Functions

- **AssertionHandler** ([StringRef](#) const &macroName, [SourceLineInfo](#) const &lineInfo, [StringRef](#) captured←  
Expression, ResultDisposition::Flags resultDisposition)
- `template<typename T >`  
`void handleExpr (ExprLhs< T > const &expr)`
- `void handleExpr (ITransientExpression const &expr)`
- `void handleMessage (ResultWas::OfType resultType, StringRef const &message)`
- `void handleExceptionThrownAsExpected ()`
- `void handleUnexpectedExceptionNotThrown ()`
- `void handleExceptionNotThrownAsExpected ()`
- `void handleThrowingCallSkipped ()`
- `void handleUnexpectedInflightException ()`
- `void complete ()`
- `void setCompleted ()`
- `auto allowThrows () const -> bool`

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.7 Catch::AssertionInfo Struct Reference

### Public Attributes

- [StringRef](#) macroName
- [SourceLineInfo](#) lineInfo
- [StringRef](#) capturedExpression
- ResultDisposition::Flags resultDisposition

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.8 Catch::AssertionReaction Struct Reference

### Public Attributes

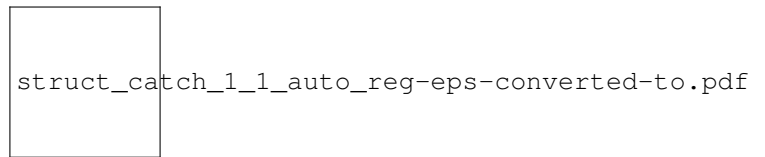
- `bool shouldDebugBreak = false`
- `bool shouldThrow = false`

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.9 Catch::AutoReg Struct Reference

Inheritance diagram for Catch::AutoReg:



### Public Member Functions

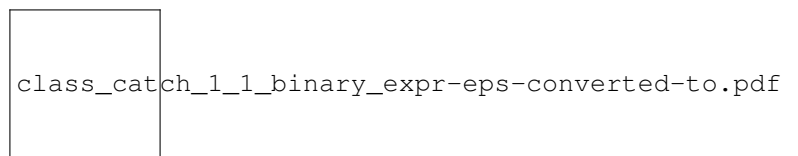
- **AutoReg** (ITestInvoker \*invoker, SourceLineInfo const &lineInfo, StringRef const &classOrMethod, NameAndTags const &nameAndTags) noexcept

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.10 Catch::BinaryExpr< LhsT, RhsT > Class Template Reference

Inheritance diagram for Catch::BinaryExpr< LhsT, RhsT >:



### Public Member Functions

- **BinaryExpr** (bool comparisonResult, LhsT lhs, StringRef op, RhsT rhs)
- template<typename T >  
  **auto operator&&** (T) const -> BinaryExpr< LhsT, RhsT const & > const
- template<typename T >  
  **auto operator||** (T) const -> BinaryExpr< LhsT, RhsT const & > const
- template<typename T >  
  **auto operator==** (T) const -> BinaryExpr< LhsT, RhsT const & > const
- template<typename T >  
  **auto operator!=** (T) const -> BinaryExpr< LhsT, RhsT const & > const
- template<typename T >  
  **auto operator>** (T) const -> BinaryExpr< LhsT, RhsT const & > const
- template<typename T >  
  **auto operator<** (T) const -> BinaryExpr< LhsT, RhsT const & > const
- template<typename T >  
  **auto operator>=** (T) const -> BinaryExpr< LhsT, RhsT const & > const
- template<typename T >  
  **auto operator<=** (T) const -> BinaryExpr< LhsT, RhsT const & > const

## Public Member Functions inherited from [Catch::ITransientExpression](#)

- [auto](#) [isBinaryExpression](#) () [const](#) -> [bool](#)
- [auto](#) [getResult](#) () [const](#) -> [bool](#)
- [ITransientExpression](#) ([bool](#) isBinaryExpression, [bool](#) result)

### Additional Inherited Members

## Public Attributes inherited from [Catch::ITransientExpression](#)

- [bool](#) [m\\_isBinaryExpression](#)
- [bool](#) [m\\_result](#)

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.11 Catch::Capturer Class Reference

### Public Member Functions

- [Capturer](#) ([StringRef](#) macroName, [SourceLineInfo](#) [const](#) &lineInfo, ResultWas::OfType [resultType](#), [StringRef](#) names)
- [void](#) [captureValue](#) ([size\\_t](#) index, [std::string](#) [const](#) &value)
- [template](#)<[typename](#) T >  
[void](#) [captureValues](#) ([size\\_t](#) index, T [const](#) &value)
- [template](#)<[typename](#) T , [typename](#)... Ts>  
[void](#) [captureValues](#) ([size\\_t](#) index, T [const](#) &value, Ts [const](#) &... values)

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.12 Catch::Matchers::StdString::CasedString Struct Reference

### Public Member Functions

- [CasedString](#) ([std::string](#) [const](#) &str, CaseSensitive::Choice [caseSensitivity](#))
- [std::string](#) [adjustString](#) ([std::string](#) [const](#) &str) [const](#)
- [std::string](#) [caseSensitivitySuffix](#) () [const](#)

### Public Attributes

- CaseSensitive::Choice [m\\_caseSensitivity](#)
- [std::string](#) [m\\_str](#)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.13 `Catch::CaseSensitive` Struct Reference

### Public Types

- enum `Choice` { `Yes` , `No` }

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

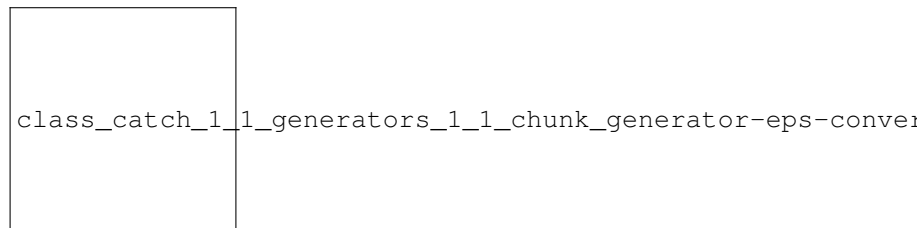
## 4.14 `Catch_global_namespace_dummy` Struct Reference

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.15 `Catch::Generators::ChunkGenerator< T >` Class Template Reference

Inheritance diagram for `Catch::Generators::ChunkGenerator< T >`:



### Public Member Functions

- `ChunkGenerator` (`size_t` size, `GeneratorWrapper< T >` generator)
- `std::vector< T >` `const & get ()` `const` override
- `bool next ()` `override`

### Additional Inherited Members

Public Types inherited from `Catch::Generators::IGenerator< std::vector< T > >`

- `using` type

### 4.15.1 Member Function Documentation

#### 4.15.1.1 get()

```
template<typename T >
std::vector< T > const & Catch::Generators::ChunkGenerator< T >::get ( ) const [inline],
[override], [virtual]
```

Implements [Catch::Generators::IGenerator< std::vector< T > >](#).

#### 4.15.1.2 next()

```
template<typename T >
bool Catch::Generators::ChunkGenerator< T >::next ( ) [inline], [override], [virtual]
```

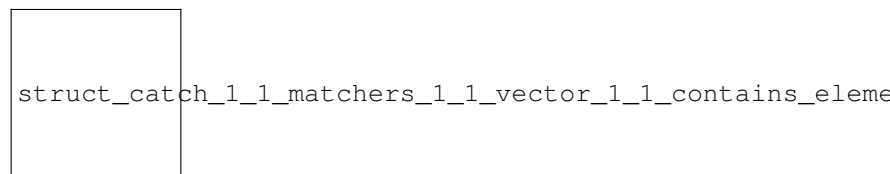
Implements [Catch::Generators::GeneratorUntypedBase](#).

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.16 Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc > Struct Template Reference

Inheritance diagram for [Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc >](#):



### Public Member Functions

- [ContainsElementMatcher](#) (T const &comparator)
- [bool match](#) (std::vector< T, Alloc > const &v) const override
- [std::string describe](#) () const override

### Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)

- [MatchAllOf< T > operator&&](#) (MatcherBase const &other) const
- [MatchAnyOf< T > operator||](#) (MatcherBase const &other) const
- [MatchNotOf< T > operator!](#) () const

## Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- [MatcherUntypedBase](#) ([MatcherUntypedBase](#) const &)=default
- [MatcherUntypedBase](#) & [operator=](#) ([MatcherUntypedBase](#) const &)=delete
- [std::string](#) [toString](#) () const

## Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- [virtual bool](#) [match](#) ([T](#) const &[arg](#)) const=0

## Public Attributes

- [T](#) const & [m\\_comparator](#)

## Additional Inherited Members

## Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- [std::string](#) [m\\_cachedToString](#)

## 4.16.1 Member Function Documentation

### 4.16.1.1 [describe\(\)](#)

```
template<typename T , typename Alloc >
std::string Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc >::describe ( ) const
[inline], [override], [virtual]
```

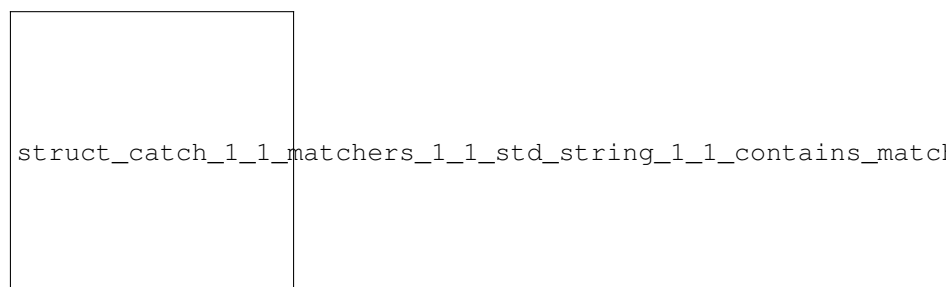
Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

The documentation for this struct was generated from the following file:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h](#)

## 4.17 [Catch::Matchers::StdString::ContainsMatcher](#) Struct Reference

Inheritance diagram for [Catch::Matchers::StdString::ContainsMatcher](#):





### Public Member Functions

- **ContainsMatcher** ([CasedString const &comparator](#))
- **bool match** ([std::string const &source](#)) [const override](#)

### Public Member Functions inherited from [Catch::Matchers::StdString::StringMatcherBase](#)

- **StringMatcherBase** ([std::string const &operation](#), [CasedString const &comparator](#))
- [std::string describe](#) () [const override](#)

### Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)

- [MatchAllOf< T > operator&&](#) ([MatcherBase const &other](#)) [const](#)
- [MatchAnyOf< T > operator||](#) ([MatcherBase const &other](#)) [const](#)
- [MatchNotOf< T > operator!](#) () [const](#)

### Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- **MatcherUntypedBase** ([MatcherUntypedBase const &](#))=default
- [MatcherUntypedBase & operator=](#) ([MatcherUntypedBase const &](#))=delete
- [std::string toString](#) () [const](#)

### Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- [virtual bool match](#) ([T const &arg](#)) [const=0](#)

### Additional Inherited Members

### Public Attributes inherited from [Catch::Matchers::StdString::StringMatcherBase](#)

- [CasedString m\\_comparator](#)
- [std::string m\\_operation](#)

### Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

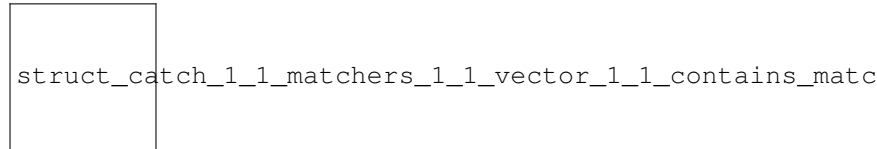
- [std::string m\\_cachedToString](#)

The documentation for this struct was generated from the following file:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h](#)

## 4.18 Catch::Matchers::Vector::ContainsMatcher< T, AllocComp, AllocMatch > Struct Template Reference

Inheritance diagram for Catch::Matchers::Vector::ContainsMatcher< T, AllocComp, AllocMatch >:



### Public Member Functions

- **ContainsMatcher** (std::vector< T, AllocComp > const &comparator)
- **bool match** (std::vector< T, AllocMatch > const &v) const override
- std::string **describe** () const override

### Public Member Functions inherited from Catch::Matchers::Impl::MatcherBase< T >

- **MatchAllOf**< T > **operator&&** (MatcherBase const &other) const
- **MatchAnyOf**< T > **operator||** (MatcherBase const &other) const
- **MatchNotOf**< T > **operator!** () const

### Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- **MatcherUntypedBase** & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

### Public Member Functions inherited from Catch::Matchers::Impl::MatcherMethod< T >

- **virtual bool match** (T const &arg) const=0

### Public Attributes

- std::vector< T, AllocComp > const & **m\_comparator**

### Additional Inherited Members

### Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

- std::string **m\_cachedToString**

### 4.18.1 Member Function Documentation

#### 4.18.1.1 describe()

```
template<typename T , typename AllocComp , typename AllocMatch >
std::string Catch::Matchers::Vector::ContainsMatcher< T, AllocComp, AllocMatch >::describe ( )
const [inline], [override], [virtual]
```

Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

The documentation for this struct was generated from the following file:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h](#)

## 4.19 Catch::Counts Struct Reference

### Public Member Functions

- [Counts operator-](#) ([Counts const](#) &[other](#)) [const](#)
- [Counts & operator+=](#) ([Counts const](#) &[other](#))
- [std::size\\_t total](#) () [const](#)
- [bool allPassed](#) () [const](#)
- [bool allOk](#) () [const](#)

### Public Attributes

- [std::size\\_t passed](#) = 0
- [std::size\\_t failed](#) = 0
- [std::size\\_t failedButOk](#) = 0

The documentation for this struct was generated from the following file:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h](#)

## 4.20 Catch::Decomposer Struct Reference

### Public Member Functions

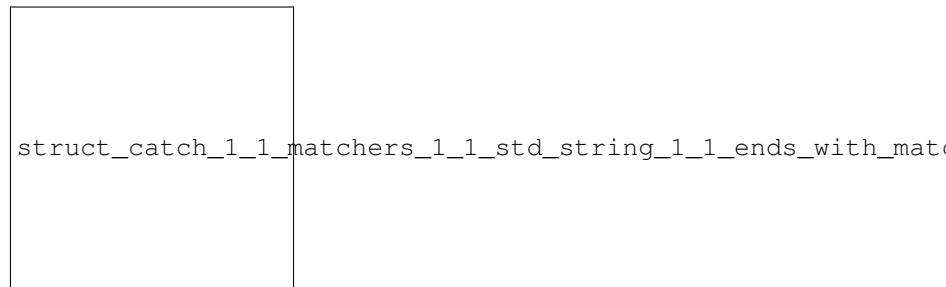
- [template<typename T >](#)  
[auto operator<=](#) ([T const](#) &[lhs](#)) -> [ExprLhs< T const & >](#)
- [auto operator<=](#) ([bool value](#)) -> [ExprLhs< bool >](#)

The documentation for this struct was generated from the following file:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h](#)

## 4.21 Catch::Matchers::StdString::EndsWithMatcher Struct Reference

Inheritance diagram for Catch::Matchers::StdString::EndsWithMatcher:



### Public Member Functions

- **EndsWithMatcher** ([CasedString](#) const &comparator)
- **bool match** (std::string const &source) const override

### Public Member Functions inherited from [Catch::Matchers::StdString::StringMatcherBase](#)

- **StringMatcherBase** (std::string const &operation, [CasedString](#) const &comparator)
- std::string **describe** () const override

### Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)

- **MatchAllOf**< T > **operator&&** ([MatcherBase](#) const &other) const
- **MatchAnyOf**< T > **operator||** ([MatcherBase](#) const &other) const
- **MatchNotOf**< T > **operator!** () const

### Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- **MatcherUntypedBase** ([MatcherUntypedBase](#) const &)=default
- **MatcherUntypedBase & operator=** ([MatcherUntypedBase](#) const &)=delete
- std::string **toString** () const

### Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- **virtual bool match** (T const &arg) const=0

### Additional Inherited Members

### Public Attributes inherited from [Catch::Matchers::StdString::StringMatcherBase](#)

- [CasedString](#) m\_comparator
- std::string m\_operation

## Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- `std::string m_cachedToString`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.22 Catch::Detail::EnumInfo Struct Reference

### Public Member Functions

- [StringRef](#) `lookup` (`int` value) `const`

### Public Attributes

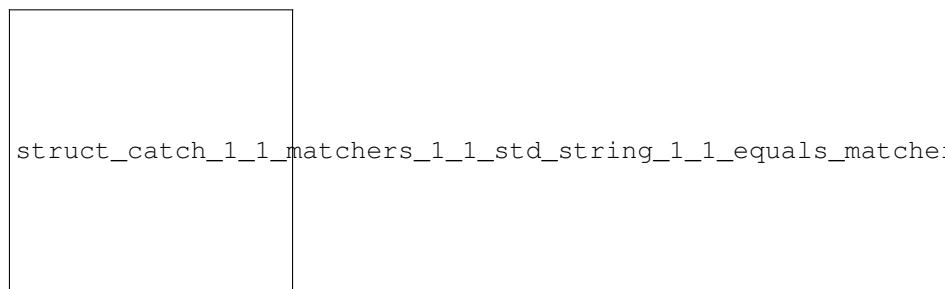
- [StringRef](#) `m_name`
- `std::vector< std::pair< int, StringRef > > m_values`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.23 Catch::Matchers::StdString::EqualsMatcher Struct Reference

Inheritance diagram for `Catch::Matchers::StdString::EqualsMatcher`:



### Public Member Functions

- `EqualsMatcher` (`CasedString` const &comparator)
- `bool match` (`std::string` const &source) `const` override

### Public Member Functions inherited from [Catch::Matchers::StdString::StringMatcherBase](#)

- `StringMatcherBase` (`std::string` const &operation, `CasedString` const &comparator)
- `std::string describe` () `const` override

### Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)

- [MatchAllOf< T > operator&&](#) ([MatcherBase const &other](#)) [const](#)
- [MatchAnyOf< T > operator||](#) ([MatcherBase const &other](#)) [const](#)
- [MatchNotOf< T > operator!](#) () [const](#)

### Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- [MatcherUntypedBase](#) ([MatcherUntypedBase const &](#))=default
- [MatcherUntypedBase & operator=](#) ([MatcherUntypedBase const &](#))=delete
- [std::string toString](#) () [const](#)

### Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- [virtual bool match](#) ([T const &arg](#)) [const=0](#)

### Additional Inherited Members

### Public Attributes inherited from [Catch::Matchers::StdString::StringMatcherBase](#)

- [CasedString m\\_comparator](#)
- [std::string m\\_operation](#)

### Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

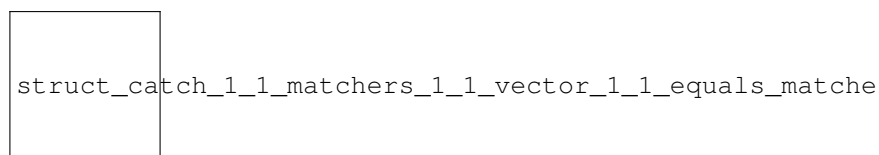
- [std::string m\\_cachedToString](#)

The documentation for this struct was generated from the following file:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h](#)

## 4.24 [Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch >](#) Struct Template Reference

Inheritance diagram for [Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch >](#):



### Public Member Functions

- [EqualsMatcher](#) ([std::vector< T, AllocComp > const &comparator](#))
- [bool match](#) ([std::vector< T, AllocMatch > const &v](#)) [const override](#)
- [std::string describe](#) () [const override](#)

**Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)**

- [MatchAllOf< T > operator&& \(MatcherBase const &other\) const](#)
- [MatchAnyOf< T > operator|| \(MatcherBase const &other\) const](#)
- [MatchNotOf< T > operator! \(\) const](#)

**Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)**

- [MatcherUntypedBase \(MatcherUntypedBase const &\)=default](#)
- [MatcherUntypedBase & operator= \(MatcherUntypedBase const &\)=delete](#)
- [std::string toString \(\) const](#)

**Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)**

- [virtual bool match \(T const &arg\) const=0](#)

**Public Attributes**

- [std::vector< T, AllocComp > const & m\\_comparator](#)

**Additional Inherited Members****Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)**

- [std::string m\\_cachedToString](#)

**4.24.1 Member Function Documentation****4.24.1.1 describe()**

```
template<typename T , typename AllocComp , typename AllocMatch >
std::string Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch >::describe ( )
const [inline], [override], [virtual]
```

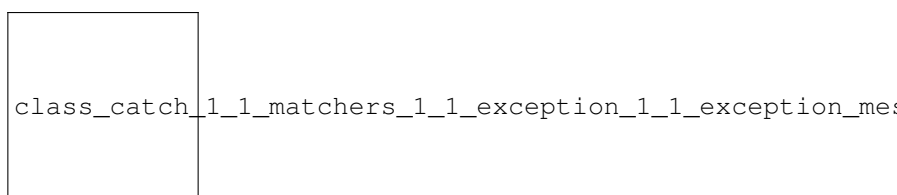
Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

The documentation for this struct was generated from the following file:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h](#)

**4.25 Catch::Matchers::Exception::ExceptionMessageMatcher Class Reference**

Inheritance diagram for [Catch::Matchers::Exception::ExceptionMessageMatcher](#):



### Public Member Functions

- **ExceptionMessageMatcher** (std::string const &message)
- **bool match** (std::exception const &ex) const override
- std::string **describe** () const override

### Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)

- **MatchAllOf< T > operator&&** (MatcherBase const &other) const
- **MatchAnyOf< T > operator||** (MatcherBase const &other) const
- **MatchNotOf< T > operator!** () const

### Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- **MatcherUntypedBase & operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

### Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- **virtual bool match** (T const &arg) const=0

### Additional Inherited Members

### Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- std::string **m\_cachedToString**

## 4.25.1 Member Function Documentation

### 4.25.1.1 describe()

```
std::string Catch::Matchers::Exception::ExceptionMessageMatcher::describe ( ) const [override],
[virtual]
```

Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.26 Catch::ExceptionTranslatorRegistrar Class Reference

### Public Member Functions

- **template<typename T >**  
**ExceptionTranslatorRegistrar** (std::string(\*translateFunction)(T &))

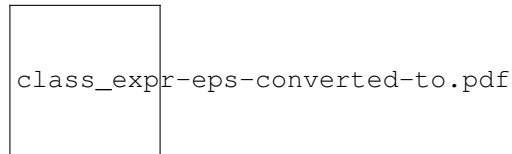
The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h



## 4.27 Expr Class Reference

Inheritance diagram for Expr:



### Public Member Functions

- virtual bool [equals](#) ([Expr](#) \*e)=0
- virtual int [interp](#) ()=0
- virtual bool [has\\_variable](#) ()=0
- virtual [Expr](#) \* [subst](#) (string varName, [Expr](#) \*replacement)=0
- virtual void [print](#) (ostream &os)=0
- string [to\\_string](#) ()
- void [pretty\\_print](#) (ostream &ostream)
- virtual void [pretty\\_print\\_at](#) (ostream &ot, precedence\_t prec)
- string [to\\_pretty\\_string](#) ()

### 4.27.1 Member Function Documentation

#### 4.27.1.1 [equals\(\)](#)

```
virtual bool Expr::equals (
    Expr * e ) [pure virtual]
```

Implemented in [Num](#), [Var](#), [Add](#), and [Mult](#).

#### 4.27.1.2 [has\\_variable\(\)](#)

```
virtual bool Expr::has_variable ( ) [pure virtual]
```

Implemented in [Num](#), [Var](#), [Add](#), and [Mult](#).

#### 4.27.1.3 [interp\(\)](#)

```
virtual int Expr::interp ( ) [pure virtual]
```

Implemented in [Num](#), [Var](#), [Add](#), and [Mult](#).

#### 4.27.1.4 [pretty\\_print\\_at\(\)](#)

```
void Expr::pretty_print_at (
    ostream & ot,
    precedence_t prec ) [virtual]
```

Reimplemented in [Add](#), and [Mult](#).

#### 4.27.1.5 print()

```
virtual void Expr::print (
    ostream & os ) [pure virtual]
```

Implemented in [Num](#), [Var](#), [Add](#), and [Mult](#).

#### 4.27.1.6 subst()

```
virtual Expr * Expr::subst (
    string varName,
    Expr * replacement ) [pure virtual]
```

Implemented in [Num](#), [Var](#), [Add](#), and [Mult](#).

The documentation for this class was generated from the following files:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.h](#)
- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.cpp](#)

## 4.28 Catch::ExprLhs< LhsT > Class Template Reference

### Public Member Functions

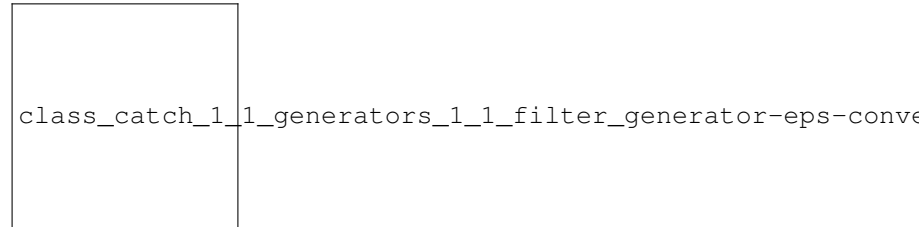
- **ExprLhs** ([LhsT](#) lhs)
- [template<typename RhsT > auto operator==\(RhsT const &rhs\) -> BinaryExpr< LhsT, RhsT const & > const](#)
- [auto operator==\(bool rhs\) -> BinaryExpr< LhsT, bool > const](#)
- [template<typename RhsT > auto operator!=\(RhsT const &rhs\) -> BinaryExpr< LhsT, RhsT const & > const](#)
- [auto operator!=\(bool rhs\) -> BinaryExpr< LhsT, bool > const](#)
- [template<typename RhsT > auto operator>\(RhsT const &rhs\) -> BinaryExpr< LhsT, RhsT const & > const](#)
- [template<typename RhsT > auto operator<\(RhsT const &rhs\) -> BinaryExpr< LhsT, RhsT const & > const](#)
- [template<typename RhsT > auto operator>=\(RhsT const &rhs\) -> BinaryExpr< LhsT, RhsT const & > const](#)
- [template<typename RhsT > auto operator<=\(RhsT const &rhs\) -> BinaryExpr< LhsT, RhsT const & > const](#)
- [template<typename RhsT > auto operator|\(RhsT const &rhs\) -> BinaryExpr< LhsT, RhsT const & > const](#)
- [template<typename RhsT > auto operator&\(RhsT const &rhs\) -> BinaryExpr< LhsT, RhsT const & > const](#)
- [template<typename RhsT > auto operator^\(RhsT const &rhs\) -> BinaryExpr< LhsT, RhsT const & > const](#)
- [template<typename RhsT > auto operator&&\(RhsT const &\) -> BinaryExpr< LhsT, RhsT const & > const](#)
- [template<typename RhsT > auto operator|| \(RhsT const &\) -> BinaryExpr< LhsT, RhsT const & > const](#)
- [auto makeUnaryExpr \(\) const -> UnaryExpr< LhsT >](#)

The documentation for this class was generated from the following file:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h](#)

## 4.29 Catch::Generators::FilterGenerator< T, Predicate > Class Template Reference

Inheritance diagram for Catch::Generators::FilterGenerator< T, Predicate >:



### Public Member Functions

- `template<typename P = Predicate>`  
**FilterGenerator** (P &&pred, GeneratorWrapper< T > &&generator)
- `T const & get () const override`
- `bool next () override`

### Additional Inherited Members

### Public Types inherited from [Catch::Generators::IGenerator< T >](#)

- `using type = T`

## 4.29.1 Member Function Documentation

### 4.29.1.1 get()

```

template<typename T , typename Predicate >
T const & Catch::Generators::FilterGenerator< T, Predicate >::get ( ) const [inline], [override],
[virtual]
  
```

Implements [Catch::Generators::IGenerator< T >](#).

### 4.29.1.2 next()

```

template<typename T , typename Predicate >
bool Catch::Generators::FilterGenerator< T, Predicate >::next ( ) [inline], [override], [virtual]
  
```

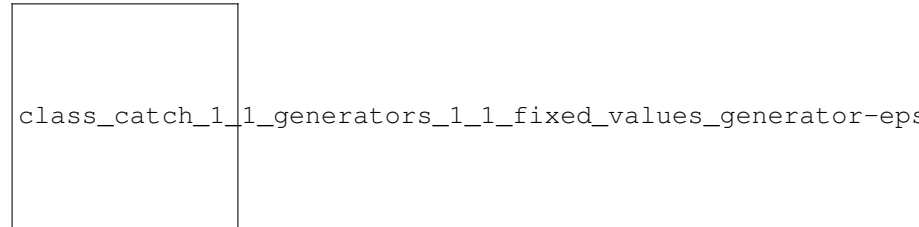
Implements [Catch::Generators::GeneratorUntypedBase](#).

The documentation for this class was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.30 `Catch::Generators::FixedValuesGenerator< T >` Class Template Reference

Inheritance diagram for `Catch::Generators::FixedValuesGenerator< T >`:



### Public Member Functions

- **FixedValuesGenerator** (`std::initializer_list< T >` values)
- `T const & get () const` [override](#)
- `bool next ()` [override](#)

### Additional Inherited Members

### Public Types inherited from `Catch::Generators::IGenerator< T >`

- `using type = T`

## 4.30.1 Member Function Documentation

### 4.30.1.1 `get()`

```
template<typename T >
T const & Catch::Generators::FixedValuesGenerator< T >::get ( ) const [inline], [override],
[virtual]
```

Implements `Catch::Generators::IGenerator< T >`.

### 4.30.1.2 `next()`

```
template<typename T >
bool Catch::Generators::FixedValuesGenerator< T >::next ( ) [inline], [override], [virtual]
```

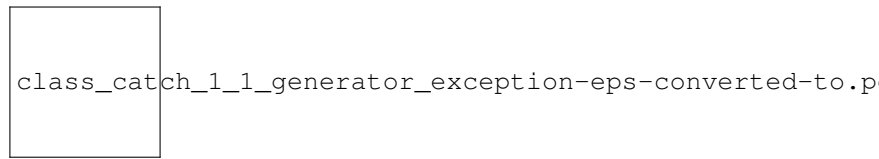
Implements `Catch::Generators::GeneratorUntypedBase`.

The documentation for this class was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.31 Catch::GeneratorException Class Reference

Inheritance diagram for Catch::GeneratorException:



### Public Member Functions

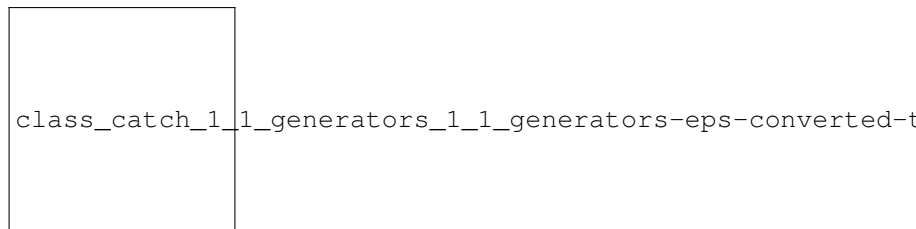
- **GeneratorException** ([const char \\*msg](#))
- [const char \\* what \(\) const noexcept override final](#)

The documentation for this class was generated from the following file:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h](#)

## 4.32 Catch::Generators::Generators< T > Class Template Reference

Inheritance diagram for Catch::Generators::Generators< T >:



### Public Member Functions

- [template<typename... Gs> Generators \(Gs &&... \[moreGenerators\]\(#\)\)](#)
- [T const & get \(\) const override](#)
- [bool next \(\) override](#)

### Additional Inherited Members

Public Types inherited from [Catch::Generators::IGenerator< T >](#)

- [using type = T](#)

### 4.32.1 Member Function Documentation

#### 4.32.1.1 `get()`

```
template<typename T >
T const & Catch::Generators::Generators< T >::get ( ) const [inline], [override], [virtual]
```

Implements [Catch::Generators::IGenerator< T >](#).

#### 4.32.1.2 `next()`

```
template<typename T >
bool Catch::Generators::Generators< T >::next ( ) [inline], [override], [virtual]
```

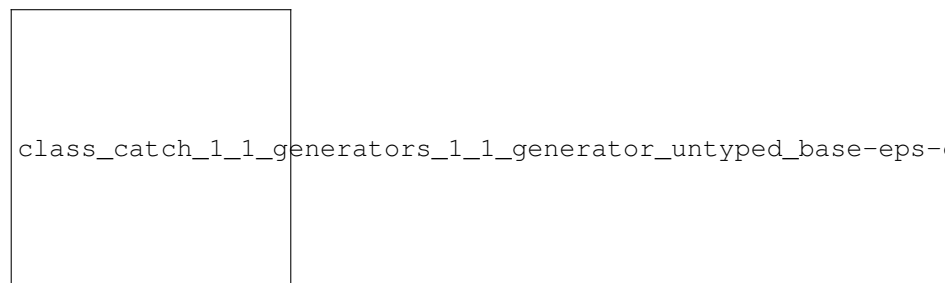
Implements [Catch::Generators::GeneratorUntypedBase](#).

The documentation for this class was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.33 `Catch::Generators::GeneratorUntypedBase` Class Reference

Inheritance diagram for `Catch::Generators::GeneratorUntypedBase`:



### Public Member Functions

- `virtual bool next ()=0`

The documentation for this class was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.34 Catch::Generators::GeneratorWrapper< T > Class Template Reference

### Public Member Functions

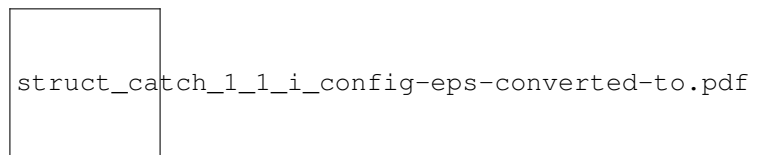
- **GeneratorWrapper** (std::unique\_ptr< IGenerator< T > > generator)
- T const & **get** () const
- bool **next** ()

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.35 Catch::IConfig Struct Reference

Inheritance diagram for Catch::IConfig:



### Public Member Functions

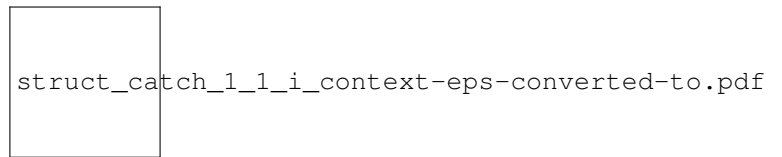
- virtual bool **allowThrows** () const =0
- virtual std::ostream & **stream** () const =0
- virtual std::string **name** () const =0
- virtual bool **includeSuccessfulResults** () const =0
- virtual bool **shouldDebugBreak** () const =0
- virtual bool **warnAboutMissingAssertions** () const =0
- virtual bool **warnAboutNoTests** () const =0
- virtual int **abortAfter** () const =0
- virtual bool **showInvisibles** () const =0
- virtual ShowDurations::OrNot **showDurations** () const =0
- virtual double **minDuration** () const =0
- virtual TestSpec const & **testSpec** () const =0
- virtual bool **hasTestFilters** () const =0
- virtual std::vector< std::string > const & **getTestsOrTags** () const =0
- virtual RunTests::InWhatOrder **runOrder** () const =0
- virtual unsigned int **rngSeed** () const =0
- virtual UseColour::YesOrNo **useColour** () const =0
- virtual std::vector< std::string > const & **getSectionsToRun** () const =0
- virtual Verbosity **verbosity** () const =0
- virtual bool **benchmarkNoAnalysis** () const =0
- virtual int **benchmarkSamples** () const =0
- virtual double **benchmarkConfidenceInterval** () const =0
- virtual unsigned int **benchmarkResamples** () const =0
- virtual std::chrono::milliseconds **benchmarkWarmupTime** () const =0

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.36 Catch::IContext Struct Reference

Inheritance diagram for Catch::IContext:



### Public Member Functions

- `virtual IResultCapture * getResultCapture ()=0`
- `virtual IRunner * getRunner ()=0`
- `virtual IConfigPtr const & getConfig () const =0`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.37 Catch::IExceptionTranslator Struct Reference

### Public Member Functions

- `virtual std::string translate (ExceptionTranslators::const_iterator it, ExceptionTranslators::const_iterator itEnd) const =0`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.38 Catch::IExceptionTranslatorRegistry Struct Reference

### Public Member Functions

- `virtual std::string translateActiveException () const =0`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`



## 4.39 Catch::Generators::!Generator< T > Struct Template Reference

Inheritance diagram for Catch::Generators::!Generator< T >:



struct\_catch\_1\_1\_generators\_1\_1\_i\_generator-eps-converted-to.pdf

### Public Types

- `using type = T`

### Public Member Functions

- `virtual T const & get () const =0`

### Public Member Functions inherited from [Catch::Generators::GeneratorUntypedBase](#)

- `virtual bool next ()=0`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.40 Catch::!GeneratorTracker Struct Reference

### Public Member Functions

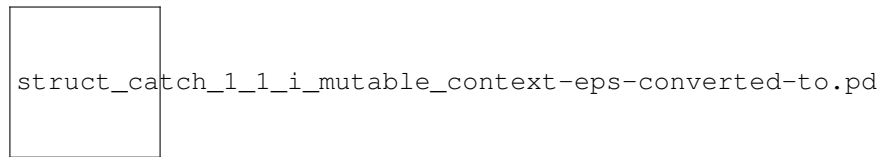
- `virtual auto hasGenerator () const -> bool=0`
- `virtual auto getGenerator () const -> Generators::GeneratorBasePtr const &=0`
- `virtual void setGenerator (Generators::GeneratorBasePtr &&generator)=0`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.41 Catch::ImmutableContext Struct Reference

Inheritance diagram for Catch::ImmutableContext:



### Public Member Functions

- `virtual void setResultCapture (IResultCapture *resultCapture)=0`
- `virtual void setRunner (IRunner *runner)=0`
- `virtual void setConfig (IConfigPtr const &config)=0`

### Public Member Functions inherited from Catch::IContext

- `virtual IResultCapture * getResultCapture ()=0`
- `virtual IRunner * getRunner ()=0`
- `virtual IConfigPtr const & getConfig () const =0`

### Friends

- `ImmutableContext & getCurrentMutableContext ()`
- `void cleanUpContext ()`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.42 Catch::ImmutableEnumValuesRegistry Struct Reference

### Public Member Functions

- `virtual Detail::EnumInfo const & registerEnum (StringRef enumName, StringRef allEnums, std::vector< int > const &values)=0`
- `template<typename E> Detail::EnumInfo const & registerEnum (StringRef enumName, StringRef allEnums, std::initializer_list< E > values)`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.43 Catch::IMutableRegistryHub Struct Reference

### Public Member Functions

- **virtual void registerReporter** (std::string const &name, IReporterFactoryPtr const &factory)=0
- **virtual void registerListener** (IReporterFactoryPtr const &factory)=0
- **virtual void registerTest** (TestCase const &testInfo)=0
- **virtual void registerTranslator** (const IExceptionTranslator \*translator)=0
- **virtual void registerTagAlias** (std::string const &alias, std::string const &tag, SourceLineInfo const &lineInfo)=0
- **virtual void registerStartupException** () noexcept=0
- **virtual IMutableEnumValuesRegistry & getMutableEnumValuesRegistry** ()=0

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.44 Catch::IRegistryHub Struct Reference

### Public Member Functions

- **virtual IReporterRegistry const & getReporterRegistry** () const =0
- **virtual ITestCaseRegistry const & getTestCaseRegistry** () const =0
- **virtual ITagAliasRegistry const & getTagAliasRegistry** () const =0
- **virtual IExceptionTranslatorRegistry const & getExceptionTranslatorRegistry** () const =0
- **virtual StartupExceptionRegistry const & getStartupExceptionRegistry** () const =0

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.45 Catch::IResultCapture Struct Reference

### Public Member Functions

- **virtual bool sectionStarted** (SectionInfo const &sectionInfo, Counts &assertions)=0
- **virtual void sectionEnded** (SectionEndInfo const &endInfo)=0
- **virtual void sectionEndedEarly** (SectionEndInfo const &endInfo)=0
- **virtual auto acquireGeneratorTracker** (StringRef generatorName, SourceLineInfo const &lineInfo) -> IGeneratorTracker &=0
- **virtual void pushScopedMessage** (MessageInfo const &message)=0
- **virtual void popScopedMessage** (MessageInfo const &message)=0
- **virtual void emplaceUnscopedMessage** (MessageBuilder const &builder)=0
- **virtual void handleFatalErrorCondition** (StringRef message)=0
- **virtual void handleExpr** (AssertionInfo const &info, ITransientExpression const &expr, AssertionReaction &reaction)=0
- **virtual void handleMessage** (AssertionInfo const &info, ResultWas::OfType resultType, StringRef const &message, AssertionReaction &reaction)=0

- `virtual void handleUnexpectedExceptionNotThrown (AssertionInfo const &info, AssertionReaction &reaction)=0`
- `virtual void handleUnexpectedInflightException (AssertionInfo const &info, std::string const &message, AssertionReaction &reaction)=0`
- `virtual void handleIncomplete (AssertionInfo const &info)=0`
- `virtual void handleNonExpr (AssertionInfo const &info, ResultWas::OfType resultType, AssertionReaction &reaction)=0`
- `virtual bool lastAssertionPassed ()=0`
- `virtual void assertionPassed ()=0`
- `virtual std::string getCurrentTestName () const =0`
- `virtual const AssertionResult * getLastResult () const =0`
- `virtual void exceptionEarlyReported ()=0`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.46 Catch::IRunner Struct Reference

### Public Member Functions

- `virtual bool aborting () const =0`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.47 Catch::is\_callable< T > Struct Template Reference

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.48 Catch::is\_callable< Fun(Args...) > Struct Template Reference

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.49 Catch::is\_callable\_tester Struct Reference

### Static Public Member Functions

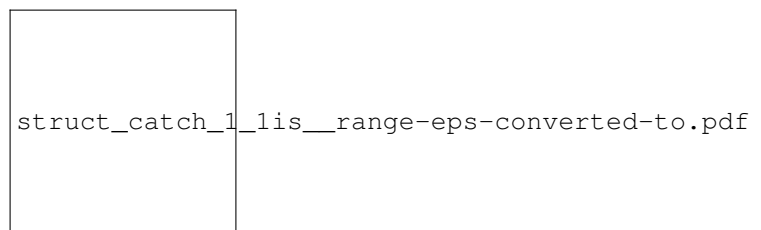
- `template<typename Fun , typename... Args>  
static true_given< decltype(std::declval< Fun >()(std::declval< Args >())...)> test (int)`
- `template<typename... >  
static std::false_type test (...)`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.50 Catch::is\_range< T > Struct Template Reference

Inheritance diagram for Catch::is\_range< T >:

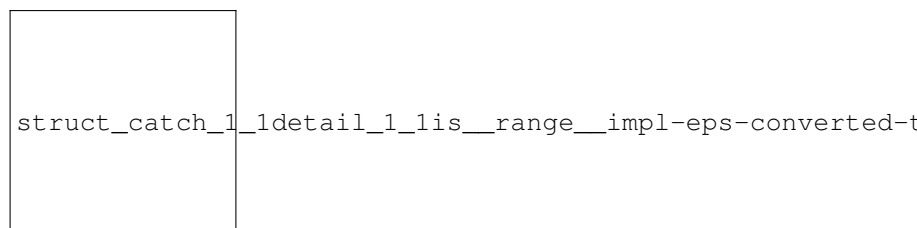


The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.51 Catch::detail::is\_range\_impl< T, typename > Struct Template Reference

Inheritance diagram for Catch::detail::is\_range\_impl< T, typename >:

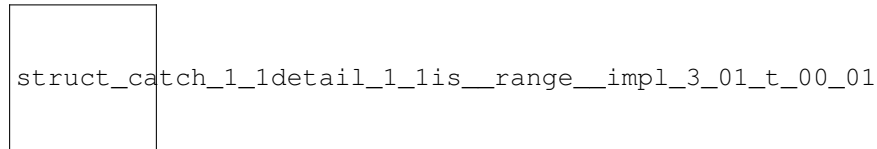


The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.52 Catch::detail::is\_range\_impl< T, typename void\_type< decltype(begin(std::declval< T >()))>::type > Struct Template Reference

Inheritance diagram for Catch::detail::is\_range\_impl< T, typename void\_type< decltype(begin(std::declval< T >()))>::type >:



The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.53 Catch::Detail::IsStreamInsertable< T > Class Template Reference

### Static Public Attributes

- `static const bool value = decltype(test<std::ostream, const T>(0))::value`

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.54 Catch::IStream Struct Reference

### Public Member Functions

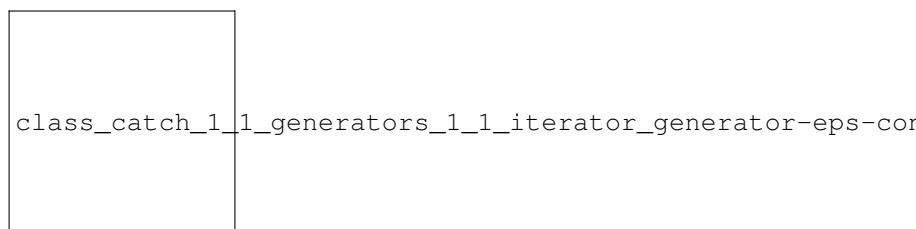
- `virtual std::ostream & stream () const =0`

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.55 Catch::Generators::IteratorGenerator< T > Class Template Reference

Inheritance diagram for Catch::Generators::IteratorGenerator< T >:



### Public Member Functions

- `template<typename InputIterator, typename InputSentinel > IteratorGenerator (InputIterator first, InputSentinel last)`
- `T const & get () const override`
- `bool next () override`

### Additional Inherited Members

### Public Types inherited from `Catch::Generators::IGenerator< T >`

- `using type = T`

## 4.55.1 Member Function Documentation

### 4.55.1.1 `get()`

```
template<typename T >
T const & Catch::Generators::IteratorGenerator< T >::get ( ) const [inline], [override], [virtual]
```

Implements `Catch::Generators::IGenerator< T >`.

### 4.55.1.2 `next()`

```
template<typename T >
bool Catch::Generators::IteratorGenerator< T >::next ( ) [inline], [override], [virtual]
```

Implements `Catch::Generators::GeneratorUntypedBase`.

The documentation for this class was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.56 Catch::ITestCaseRegistry Struct Reference

### Public Member Functions

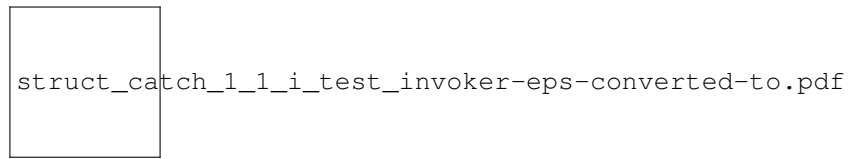
- `virtual std::vector< TestCase > const & getAllTests () const =0`
- `virtual std::vector< TestCase > const & getAllTestsSorted (IConfig const &config) const =0`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.57 Catch::ITestInvoker Struct Reference

Inheritance diagram for Catch::ITestInvoker:



### Public Member Functions

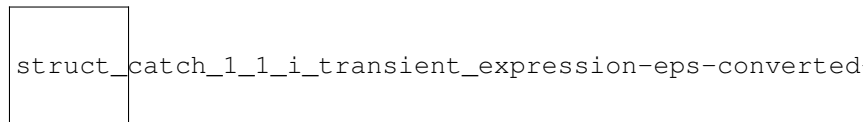
- `virtual void invoke () const =0`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.58 Catch::ITransientExpression Struct Reference

Inheritance diagram for Catch::ITransientExpression:



### Public Member Functions

- `auto isBinaryExpression () const -> bool`
- `auto getResult () const -> bool`
- `virtual void streamReconstructedExpression (std::ostream &os) const =0`
- `ITransientExpression (bool isBinaryExpression, bool result)`

### Public Attributes

- `bool m_isBinaryExpression`
- `bool m_result`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`



## 4.59 Catch::LazyExpression Class Reference

### Public Member Functions

- **LazyExpression** ([bool isNegated](#))
- **LazyExpression** ([LazyExpression const &other](#))
- [LazyExpression](#) & **operator=** ([LazyExpression const &](#))=delete
- **operator bool** () [const](#)

### Friends

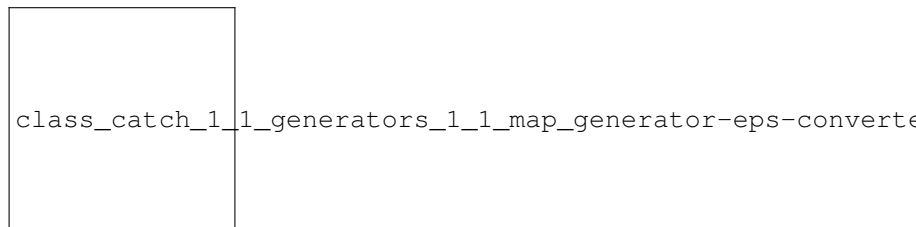
- [class](#) **AssertionHandler**
- [struct](#) **AssertionStats**
- [class](#) **RunContext**
- [auto](#) **operator<<** ([std::ostream &os](#), [LazyExpression const &lazyExpr](#)) -> [std::ostream &](#)

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.60 Catch::Generators::MapGenerator< T, U, Func > Class Template Reference

Inheritance diagram for Catch::Generators::MapGenerator< T, U, Func >:



### Public Member Functions

- [template<typename F2 = Func>](#)  
**MapGenerator** ([F2 &&function](#), [GeneratorWrapper< U > &&generator](#))
- [T const &](#) **get** () [const override](#)
- [bool](#) **next** () [override](#)

### Additional Inherited Members

### Public Types inherited from [Catch::Generators::IGenerator< T >](#)

- [using](#) **type** = [T](#)

### 4.60.1 Member Function Documentation

#### 4.60.1.1 `get()`

```
template<typename T , typename U , typename Func >
T const & Catch::Generators::MapGenerator< T, U, Func >::get ( ) const [inline], [override],
[virtual]
```

Implements [Catch::Generators::IGenerator< T >](#).

#### 4.60.1.2 `next()`

```
template<typename T , typename U , typename Func >
bool Catch::Generators::MapGenerator< T, U, Func >::next ( ) [inline], [override], [virtual]
```

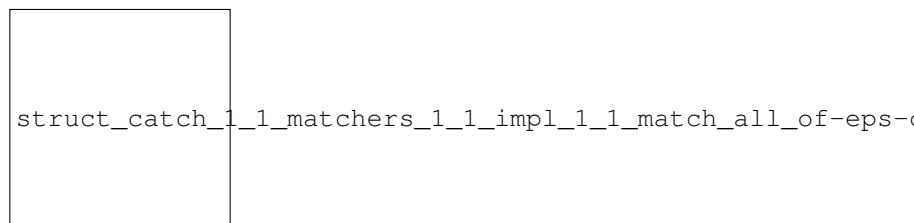
Implements [Catch::Generators::GeneratorUntypedBase](#).

The documentation for this class was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.61 `Catch::Matchers::Impl::MatchAllOf< ArgT >` Struct Template Reference

Inheritance diagram for `Catch::Matchers::Impl::MatchAllOf< ArgT >`:



### Public Member Functions

- `bool match (ArgT const &arg) const` [override](#)
- `std::string describe () const` [override](#)
- `MatchAllOf< ArgT > operator&& (MatcherBase< ArgT > const &other)`

### Public Member Functions inherited from `Catch::Matchers::Impl::MatcherBase< ArgT >`

- `MatchAllOf< ArgT > operator&& (MatcherBase const &other) const`
- `MatchAnyOf< ArgT > operator|| (MatcherBase const &other) const`
- `MatchNotOf< ArgT > operator! () const`

**Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)**

- **MatcherUntypedBase** ([MatcherUntypedBase const &](#))=default
- [MatcherUntypedBase &](#) **operator=** ([MatcherUntypedBase const &](#))=delete
- `std::string toString () const`

**Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< ObjectT >](#)**

- `virtual bool match (ObjectT const &arg) const =0`

**Public Attributes**

- `std::vector< MatcherBase< ArgT > const \* > m\_matchers`

**Additional Inherited Members****Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)**

- `std::string m_cachedToString`

**4.61.1 Member Function Documentation****4.61.1.1 describe()**

```
template<typename ArgT >
std::string Catch::Matchers::Impl::MatchAllOf< ArgT >::describe ( ) const [inline], [override],
[virtual]
```

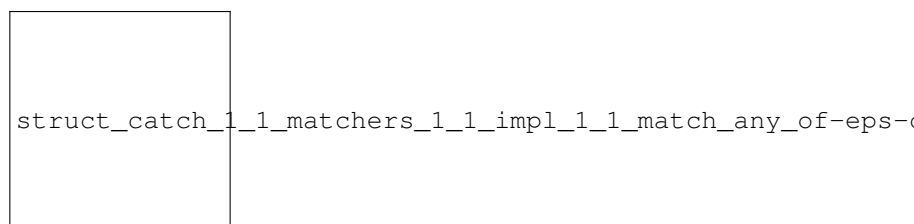
Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

**4.62 [Catch::Matchers::Impl::MatchAnyOf< ArgT >](#) Struct Template Reference**

Inheritance diagram for [Catch::Matchers::Impl::MatchAnyOf< ArgT >](#):



## Public Member Functions

- `bool match (ArgT const &arg) const` [override](#)
- `std::string describe ()` [const override](#)
- `MatchAnyOf< ArgT > operator|| (MatcherBase< ArgT > const &other)`

## Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< ArgT >](#)

- `MatchAllOf< ArgT > operator&& (MatcherBase const &other) const`
- `MatchAnyOf< ArgT > operator|| (MatcherBase const &other) const`
- `MatchNotOf< ArgT > operator! () const`

## Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- `MatcherUntypedBase (MatcherUntypedBase const &)=default`
- `MatcherUntypedBase & operator= (MatcherUntypedBase const &)=delete`
- `std::string toString ()` [const](#)

## Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< ObjectT >](#)

- `virtual bool match (ObjectT const &arg) const` `=0`

## Public Attributes

- `std::vector< MatcherBase< ArgT > const * > m_matchers`

## Additional Inherited Members

## Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- `std::string m_cachedToString`

## 4.62.1 Member Function Documentation

### 4.62.1.1 describe()

```
template<typename ArgT >
std::string Catch::Matchers::Impl::MatchAnyOf< ArgT >::describe ( ) const [inline], [override],
[virtual]
```

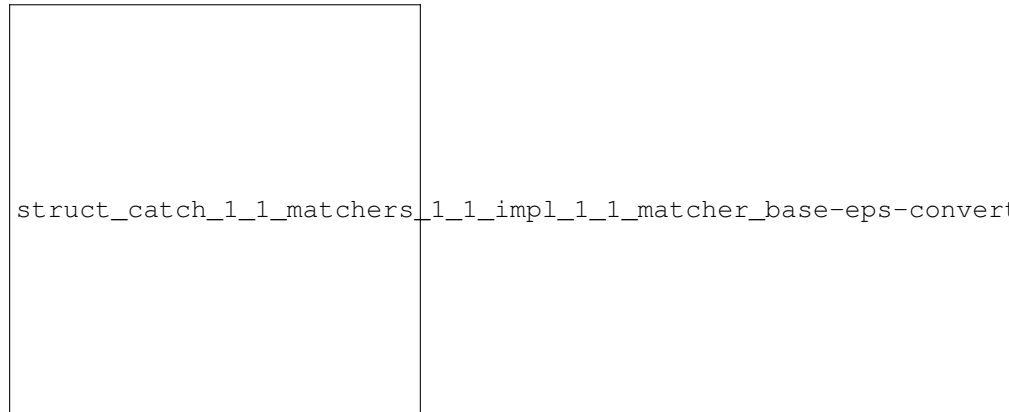
Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.63 Catch::Matchers::Impl::MatcherBase< T > Struct Template Reference

Inheritance diagram for Catch::Matchers::Impl::MatcherBase< T >:



### Public Member Functions

- [MatchAllOf< T > operator&& \(MatcherBase const &other\) const](#)
- [MatchAnyOf< T > operator|| \(MatcherBase const &other\) const](#)
- [MatchNotOf< T > operator! \(\) const](#)

### Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- [MatcherUntypedBase \(MatcherUntypedBase const &\)=default](#)
- [MatcherUntypedBase & operator= \(MatcherUntypedBase const &\)=delete](#)
- [std::string toString \(\) const](#)

### Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- [virtual bool match \(T const &arg\) const=0](#)

### Additional Inherited Members

### Protected Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- [virtual std::string describe \(\) const =0](#)

### Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

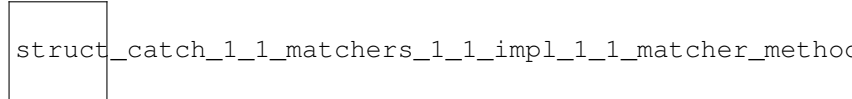
- [std::string m\\_cachedToString](#)

The documentation for this struct was generated from the following file:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h](#)

## 4.64 Catch::Matchers::Impl::MatcherMethod< ObjectT > Struct Template Reference

Inheritance diagram for Catch::Matchers::Impl::MatcherMethod< ObjectT >:



### Public Member Functions

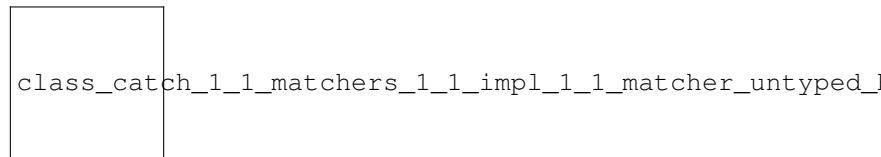
- `virtual bool match (ObjectT const &arg) const =0`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.65 Catch::Matchers::Impl::MatcherUntypedBase Class Reference

Inheritance diagram for Catch::Matchers::Impl::MatcherUntypedBase:



### Public Member Functions

- `MatcherUntypedBase (MatcherUntypedBase const &)=default`
- `MatcherUntypedBase & operator= (MatcherUntypedBase const &)=delete`
- `std::string toString () const`

### Protected Member Functions

- `virtual std::string describe () const =0`

### Protected Attributes

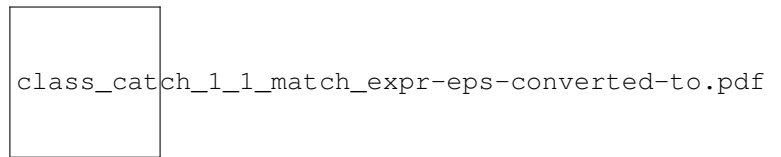
- `std::string m_cachedToString`

The documentation for this class was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.66 Catch::MatchExpr< ArgT, MatcherT > Class Template Reference

Inheritance diagram for Catch::MatchExpr< ArgT, MatcherT >:



### Public Member Functions

- **MatchExpr** ([ArgT const &arg](#), [MatcherT const &matcher](#), [StringRef const &matcherString](#))
- **void streamReconstructedExpression** ([std::ostream &os](#)) [const override](#)

### Public Member Functions inherited from [Catch::ITransientExpression](#)

- **auto isBinaryExpression** () [const -> bool](#)
- **auto getResult** () [const -> bool](#)
- **ITransientExpression** ([bool isBinaryExpression](#), [bool result](#))

### Additional Inherited Members

### Public Attributes inherited from [Catch::ITransientExpression](#)

- [bool m\\_isBinaryExpression](#)
- [bool m\\_result](#)

## 4.66.1 Member Function Documentation

### 4.66.1.1 streamReconstructedExpression()

```

template<typename ArgT , typename MatcherT >
void Catch::MatchExpr< ArgT, MatcherT >::streamReconstructedExpression (
    std::ostream & os ) const [inline], [override], [virtual]

```

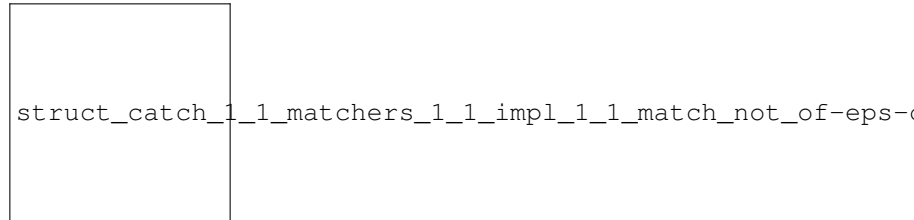
Implements [Catch::ITransientExpression](#).

The documentation for this class was generated from the following file:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h](#)

## 4.67 Catch::Matchers::Impl::MatchNotOf< ArgT > Struct Template Reference

Inheritance diagram for Catch::Matchers::Impl::MatchNotOf< ArgT >:



### Public Member Functions

- **MatchNotOf** ([MatcherBase< ArgT > const](#) &underlyingMatcher)
- **bool match** ([ArgT const](#) &arg) [const override](#)
- [std::string describe](#) () [const override](#)

### Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< ArgT >](#)

- [MatchAllOf< ArgT > operator&&](#) ([MatcherBase const](#) &other) [const](#)
- [MatchAnyOf< ArgT > operator||](#) ([MatcherBase const](#) &other) [const](#)
- [MatchNotOf< ArgT > operator!](#) () [const](#)

### Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- **MatcherUntypedBase** ([MatcherUntypedBase const](#) &)=default
- [MatcherUntypedBase](#) & **operator=** ([MatcherUntypedBase const](#) &)=delete
- [std::string toString](#) () [const](#)

### Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< ObjectT >](#)

- **virtual bool match** ([ObjectT const](#) &arg) [const](#) =0

### Public Attributes

- [MatcherBase< ArgT > const](#) & m\_underlyingMatcher

### Additional Inherited Members

### Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- [std::string m\\_cachedToString](#)



## 4.67.1 Member Function Documentation

### 4.67.1.1 describe()

```
template<typename ArgT >
std::string Catch::Matchers::Impl::MatchNotOf< ArgT >::describe ( ) const [inline], [override],
[virtual]
```

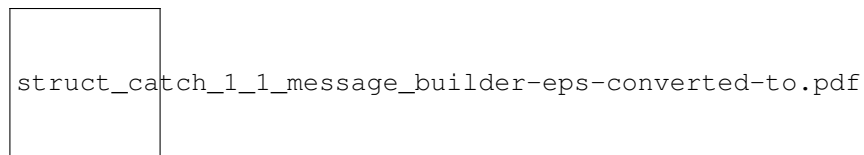
Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.68 Catch::MessageBuilder Struct Reference

Inheritance diagram for Catch::MessageBuilder:



### Public Member Functions

- **MessageBuilder** ([StringRef](#) const &macroName, [SourceLineInfo](#) const &lineInfo, ResultWas::OfType type)
- `template<typename T >`  
[MessageBuilder](#) & **operator**<< ([T](#) const &value)

### Public Member Functions inherited from [Catch::MessageStream](#)

- `template<typename T >`  
[MessageStream](#) & **operator**<< ([T](#) const &value)

### Public Attributes

- [MessageInfo](#) m\_info

### Public Attributes inherited from [Catch::MessageStream](#)

- [ReusableStringStream](#) m\_stream

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.69 Catch::MessageInfo Struct Reference

### Public Member Functions

- **MessageInfo** ([StringRef](#) const &[\\_macroName](#), [SourceLineInfo](#) const &[\\_lineInfo](#), [ResultWas::OfType](#) [\\_type](#))
- **bool operator==** ([MessageInfo](#) const &[other](#)) const
- **bool operator<** ([MessageInfo](#) const &[other](#)) const

### Public Attributes

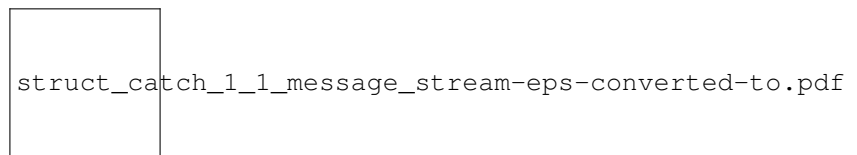
- [StringRef](#) **macroName**
- std::string **message**
- [SourceLineInfo](#) **lineInfo**
- [ResultWas::OfType](#) **type**
- unsigned int **sequence**

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.70 Catch::MessageStream Struct Reference

Inheritance diagram for Catch::MessageStream:



### Public Member Functions

- template<typename [T](#) >  
[MessageStream](#) & **operator<<** ([T](#) const &[value](#))

### Public Attributes

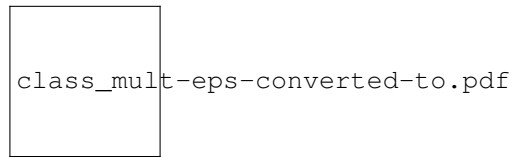
- [ReusableStringStream](#) **m\_stream**

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.71 Mult Class Reference

Inheritance diagram for Mult:



### Public Member Functions

- `Mult (Expr *lhs, Expr *rhs)`  
*Constructor for the `Mult` class.*
- `bool equals (Expr *e)`  
*Checks if this `Mult` expression is equal to another expression.*
- `int interp ()`  
*Evaluates the multiplication expression.*
- `bool has_variable ()`  
*Checks if the expression contains any variables.*
- `Expr * subst (string varName, Expr *replacement)`  
*Substitutes a variable within the expression with another expression.*
- `virtual void print (ostream &os)`  
*Prints the `Mult` expression in a human-readable form.*
- `void pretty_print_at (ostream &ot, precedence_t prec)`  
*Pretty prints the `Mult` expression with appropriate precedence.*

### Public Member Functions inherited from Expr

- `string to_string ()`
- `void pretty_print (ostream &ostream)`
- `string to_pretty_string ()`

### Public Attributes

- `Expr * lhs`
- `Expr * rhs`

### 4.71.1 Constructor & Destructor Documentation

#### 4.71.1.1 Mult()

```
Mult::Mult (
    Expr * lhs,
    Expr * rhs )
```

Constructor for the `Mult` class.

## Parameters

<i>lhs</i>	The left-hand side expression of the multiplication.
<i>rhs</i>	The right-hand side expression of the multiplication. Initializes a <a href="#">Mult</a> object with two expressions to be multiplied.

## 4.71.2 Member Function Documentation

### 4.71.2.1 equals()

```
bool Mult::equals (
    Expr * e ) [virtual]
```

Checks if this [Mult](#) expression is equal to another expression.

## Parameters

<i>e</i>	The expression to compare with.
----------	---------------------------------

## Returns

True if both lhs and rhs of [Mult](#) are equal to those of e, false otherwise.

Implements [Expr](#).

### 4.71.2.2 has\_variable()

```
bool Mult::has_variable ( ) [virtual]
```

Checks if the expression contains any variables.

## Returns

True if either lhs or rhs contains a variable, false otherwise.

Implements [Expr](#).

### 4.71.2.3 interp()

```
int Mult::interp ( ) [virtual]
```

Evaluates the multiplication expression.

## Returns

The product of the interpretations of lhs and rhs.

Implements [Expr](#).

### 4.71.2.4 pretty\_print\_at()

```
void Mult::pretty_print_at (
    ostream & ot,
    precedence_t prec ) [virtual]
```

Pretty prints the [Mult](#) expression with appropriate precedence.

## Parameters

<i>o</i>	The output stream to print to.
<i>prec</i>	The current precedence level.

Reimplemented from [Expr](#).

## 4.71.2.5 print()

```
void Mult::print (
    ostream & ostream ) [virtual]
```

Prints the [Mult](#) expression in a human-readable form.

## Parameters

<i>ostream</i>	The output stream to print to.
----------------	--------------------------------

Implements [Expr](#).

## 4.71.2.6 subst()

```
Expr * Mult::subst (
    string varName,
    Expr * replacement ) [virtual]
```

Substitutes a variable within the expression with another expression.

## Parameters

<i>varName</i>	The name of the variable to be substituted.
<i>replacement</i>	The expression to substitute in place of the variable.

## Returns

A new [Mult](#) expression with the variable substituted.

Implements [Expr](#).

The documentation for this class was generated from the following files:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.h](#)
- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.cpp](#)

## 4.72 Catch::NameAndTags Struct Reference

## Public Member Functions

- **NameAndTags** ([StringRef const](#) &name\_[\\_](#)=[StringRef\(\)](#), [StringRef const](#) &tags\_[\\_](#)=[StringRef\(\)](#)) [noexcept](#)

**Public Attributes**

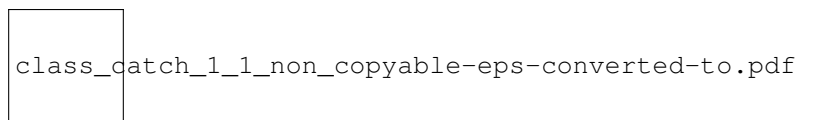
- [StringRef](#) **name**
- [StringRef](#) **tags**

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

**4.73 Catch::NonCopyable Class Reference**

Inheritance diagram for Catch::NonCopyable:



The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

**4.74 Num Class Reference**

Inheritance diagram for Num:

**Public Member Functions**

- [Num](#) (int val)  
*Constructor for [Num](#).*
- bool [equals](#) ([Expr](#) \*e)  
*Implementation of the equals function for [Num](#).*
- int [interp](#) ()  
*the [interp\(\)](#) function for [Num](#) class.*
- bool [has\\_variable](#) ()  
*the [has\\_variable\(\)](#) function for [Num](#) class.*
- [Expr](#) \* [subst](#) (string varName, [Expr](#) \*replacement)  
*The [subst\(\)](#) function for [Num](#).*
- virtual void [print](#) (ostream &os)  
*the print function for [Num](#).*
- string [to\\_string](#) ()

## Public Member Functions inherited from [Expr](#)

- string [to\\_string](#) ()
- void [pretty\\_print](#) (ostream &ostream)
- virtual void [pretty\\_print\\_at](#) (ostream &ot, precedence\_t prec)
- string [to\\_pretty\\_string](#) ()

## Public Attributes

- int [val](#)

## 4.74.1 Constructor & Destructor Documentation

### 4.74.1.1 Num()

```
Num::Num (
    int val )
```

Constructor for [Num](#).

#### Parameters

<a href="#">val</a>	The integer value of the <a href="#">Num</a> object. Creates a <a href="#">Num</a> object out of val.
---------------------	---

## 4.74.2 Member Function Documentation

### 4.74.2.1 equals()

```
bool Num::equals (
    Expr * e ) [virtual]
```

Implementation of the equals function for [Num](#).

#### Parameters

<a href="#">e</a>	the expression you compare.
-------------------	-----------------------------

#### Returns

false if num is a null pointer, true otherwise. Verifies the current [Num](#) object is equal to a different expression.

Implements [Expr](#).

### 4.74.2.2 has\_variable()

```
bool Num::has_variable ( ) [virtual]
```

the [has\\_variable\(\)](#) function for [Num](#) class.

**Returns**

ALWAYS will return false. Verifies that there are no variables.

Implements [Expr](#).

**4.74.2.3 interp()**

```
int Num::interp ( ) [virtual]
```

the [interp\(\)](#) function for [Num](#) class.

**Returns**

the integer val of [Num](#) object.

Implements [Expr](#).

**4.74.2.4 print()**

```
void Num::print (
    ostream & os ) [virtual]
```

the print function for [Num](#).

**Parameters**

<i>os</i>	The output stream to print to. Prints the value of the <a href="#">Num</a> object as a string to the specified output stream.
-----------	---

Implements [Expr](#).

**4.74.2.5 subst()**

```
Expr * Num::subst (
    string varName,
    Expr * replacement ) [virtual]
```

The [subst\(\)](#) function for [Num](#).

**Parameters**

<i>varName</i>	the variable that will be replaced.
<i>replacement</i>	The replacement expression.

**Returns**

The new expression with the variable substituted. Swaps varName with a replacement expression.



Implements [Expr](#).

The documentation for this class was generated from the following files:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.h`
- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.cpp`

## 4.75 `Catch::Option< T >` Class Template Reference

### Public Member Functions

- `Option (T const &_value)`
- `Option (Option const &_other)`
- `Option & operator= (Option const &_other)`
- `Option & operator= (T const &_value)`
- `void reset ()`
- `T & operator* ()`
- `T const & operator* () const`
- `T * operator-> ()`
- `const T * operator-> () const`
- `T valueOr (T const &defaultValue) const`
- `bool some () const`
- `bool none () const`
- `bool operator! () const`
- `operator bool () const`

The documentation for this class was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.76 `Catch::pluralise` Struct Reference

### Public Member Functions

- `pluralise (std::size_t count, std::string const &label)`

### Public Attributes

- `std::size_t m_count`
- `std::string m_label`

### Friends

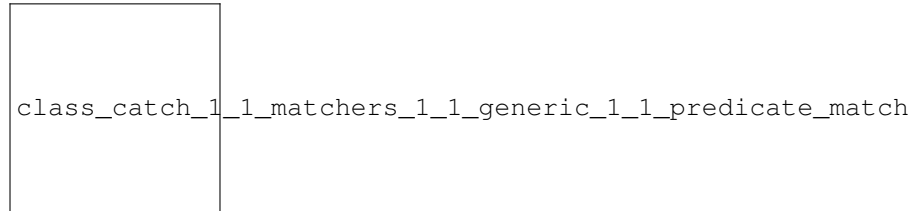
- `std::ostream & operator<< (std::ostream &os, pluralise const &pluraliser)`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.77 Catch::Matchers::Generic::PredicateMatcher< T > Class Template Reference

Inheritance diagram for Catch::Matchers::Generic::PredicateMatcher< T >:



### Public Member Functions

- **PredicateMatcher** (std::function< bool(T const &)> const &elem, std::string const &descr)
- bool match (T const &item) const override
- std::string describe () const override

### Public Member Functions inherited from Catch::Matchers::Impl::MatcherBase< T >

- MatchAllOf< T > operator&& (MatcherBase const &other) const
- MatchAnyOf< T > operator|| (MatcherBase const &other) const
- MatchNotOf< T > operator! () const

### Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- **MatcherUntypedBase** & operator= (MatcherUntypedBase const &)=delete
- std::string toString () const

### Additional Inherited Members

### Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

- std::string m\_cachedToString

## 4.77.1 Member Function Documentation

### 4.77.1.1 describe()

```

template<typename T >
std::string Catch::Matchers::Generic::PredicateMatcher< T >::describe ( ) const [inline],
[override], [virtual]
  
```

Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

### 4.77.1.2 match()

```
template<typename T >
bool Catch::Matchers::Generic::PredicateMatcher< T >::match (
    T const & item ) const [inline], [override], [virtual]
```

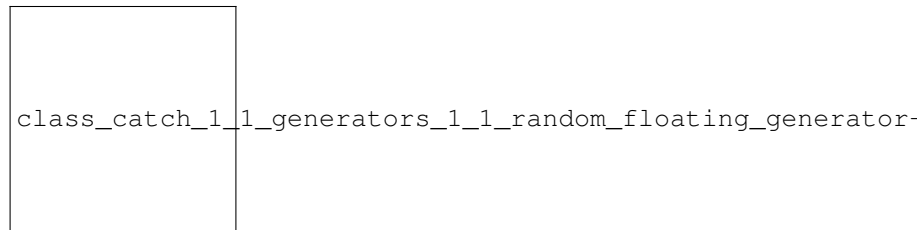
Implements [Catch::Matchers::Impl::MatcherMethod< T >](#).

The documentation for this class was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.78 Catch::Generators::RandomFloatingGenerator< Float > Class Template Reference

Inheritance diagram for `Catch::Generators::RandomFloatingGenerator< Float >`:



### Public Member Functions

- **RandomFloatingGenerator** ([Float a](#), [Float b](#))
- [Float const & get \(\) const](#) [override](#)
- [bool next \(\)](#) [override](#)

### Additional Inherited Members

### Public Types inherited from [Catch::Generators::IGenerator< Float >](#)

- [using type](#)

## 4.78.1 Member Function Documentation

### 4.78.1.1 get()

```
template<typename Float >
Float const & Catch::Generators::RandomFloatingGenerator< Float >::get ( ) const [inline],
[override], [virtual]
```

Implements [Catch::Generators::IGenerator< Float >](#).

#### 4.78.1.2 next()

```
template<typename Float >
bool Catch::Generators::RandomFloatingGenerator< Float >::next ( ) [inline], [override],
[virtual]
```

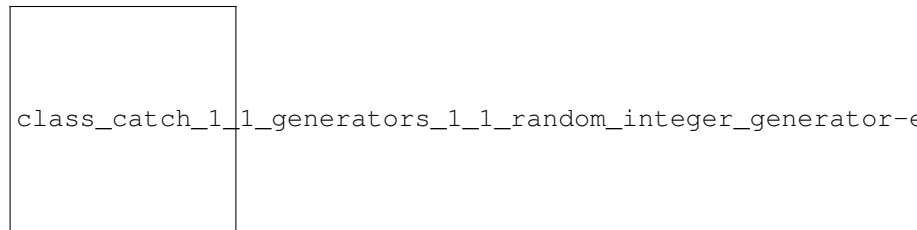
Implements [Catch::Generators::GeneratorUntypedBase](#).

The documentation for this class was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.79 Catch::Generators::RandomIntegerGenerator< Integer > Class Template Reference

Inheritance diagram for `Catch::Generators::RandomIntegerGenerator< Integer >`:



### Public Member Functions

- `RandomIntegerGenerator (Integer a, Integer b)`
- `Integer const & get () const` [override](#)
- `bool next ()` [override](#)

### Additional Inherited Members

Public Types inherited from [Catch::Generators::IGenerator< Integer >](#)

- `using type`

### 4.79.1 Member Function Documentation

#### 4.79.1.1 get()

```
template<typename Integer >
Integer const & Catch::Generators::RandomIntegerGenerator< Integer >::get ( ) const [inline],
[override], [virtual]
```

Implements [Catch::Generators::IGenerator< Integer >](#).

## 4.79.1.2 next()

```
template<typename Integer >
bool Catch::Generators::RandomIntegerGenerator< Integer >::next ( ) [inline], [override],
[virtual]
```

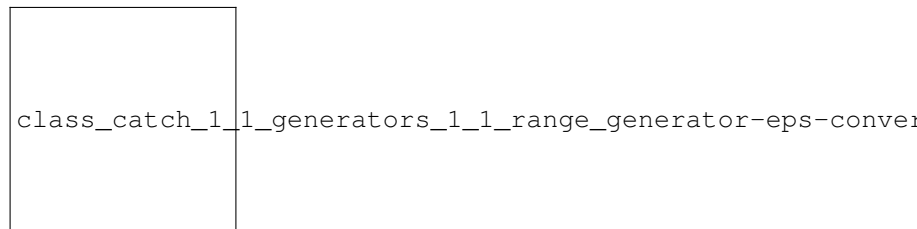
Implements [Catch::Generators::GeneratorUntypedBase](#).

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.80 Catch::Generators::RangeGenerator< T > Class Template Reference

Inheritance diagram for `Catch::Generators::RangeGenerator< T >`:



### Public Member Functions

- **RangeGenerator** (T const &start, T const &end, T const &step)
- **RangeGenerator** (T const &start, T const &end)
- T const & get () const override
- bool next () override

### Additional Inherited Members

### Public Types inherited from [Catch::Generators::IGenerator< T >](#)

- using type = T

## 4.80.1 Member Function Documentation

## 4.80.1.1 get()

```
template<typename T >
T const & Catch::Generators::RangeGenerator< T >::get ( ) const [inline], [override], [virtual]
```

Implements [Catch::Generators::IGenerator< T >](#).

### 4.80.1.2 next()

```
template<typename T >
bool Catch::Generators::RangeGenerator< T >::next ( ) [inline], [override], [virtual]
```

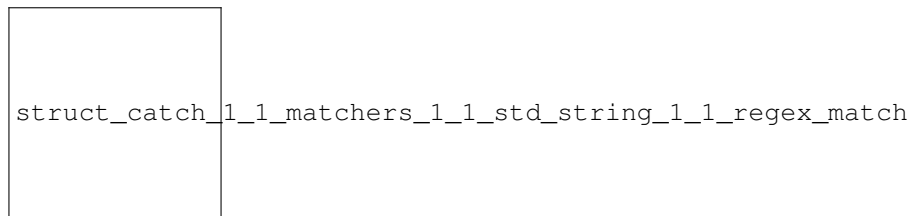
Implements [Catch::Generators::GeneratorUntypedBase](#).

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.81 Catch::Matchers::StdString::RegexMatcher Struct Reference

Inheritance diagram for Catch::Matchers::StdString::RegexMatcher:



### Public Member Functions

- **RegexMatcher** (std::string [regex](#), CaseSensitive::Choice [caseSensitivity](#))
- **bool match** (std::string [const](#) &[matchee](#)) [const override](#)
- std::string [describe](#) () [const override](#)

### Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)

- **MatchAllOf**< T > **operator&&** ([MatcherBase](#) [const](#) &[other](#)) [const](#)
- **MatchAnyOf**< T > **operator||** ([MatcherBase](#) [const](#) &[other](#)) [const](#)
- **MatchNotOf**< T > **operator!** () [const](#)

### Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- **MatcherUntypedBase** ([MatcherUntypedBase](#) [const](#) &)=default
- **MatcherUntypedBase** & **operator=** ([MatcherUntypedBase](#) [const](#) &)=delete
- std::string **toString** () [const](#)

### Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- **virtual bool match** ([T](#) [const](#) &[arg](#)) [const=0](#)

**Additional Inherited Members****Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)**

- `std::string m_cachedToString`

**4.81.1 Member Function Documentation****4.81.1.1 describe()**

```
std::string Catch::Matchers::StdString::RegexMatcher::describe ( ) const [override], [virtual]
```

Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

**4.82 Catch::RegistrarForTagAliases Struct Reference****Public Member Functions**

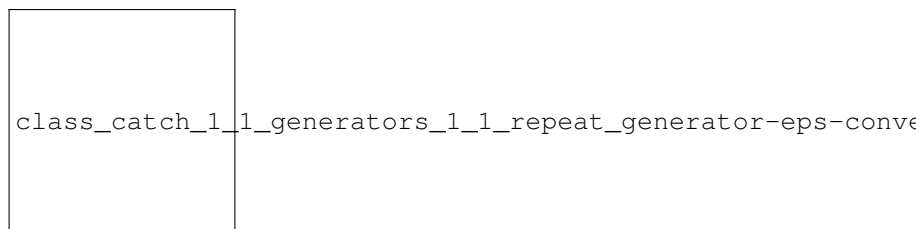
- **RegistrarForTagAliases** (`char const *alias`, `char const *tag`, `SourceLineInfo const &lineInfo`)

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

**4.83 Catch::Generators::RepeatGenerator< T > Class Template Reference**

Inheritance diagram for `Catch::Generators::RepeatGenerator< T >`:

**Public Member Functions**

- **RepeatGenerator** (`size_t repeats`, `GeneratorWrapper< T > &&generator`)
- `T const & get () const override`
- `bool next () override`

## Additional Inherited Members

### Public Types inherited from [Catch::Generators::IGenerator< T >](#)

- [using type = T](#)

## 4.83.1 Member Function Documentation

### 4.83.1.1 `get()`

```
template<typename T >
T const & Catch::Generators::RepeatGenerator< T >::get ( ) const [inline], [override], [virtual]
```

Implements [Catch::Generators::IGenerator< T >](#).

### 4.83.1.2 `next()`

```
template<typename T >
bool Catch::Generators::RepeatGenerator< T >::next ( ) [inline], [override], [virtual]
```

Implements [Catch::Generators::GeneratorUntypedBase](#).

The documentation for this class was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.84 [Catch::ResultDisposition](#) Struct Reference

### Public Types

- enum **Flags** { **Normal** = 0x01 , **ContinueOnFailure** = 0x02 , **FalseTest** = 0x04 , **SuppressFail** = 0x08 }

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.85 [Catch::ResultWas](#) Struct Reference

### Public Types

- enum **OfType** {  
**Unknown** = -1 , **Ok** = 0 , **Info** = 1 , **Warning** = 2 ,  
**FailureBit** = 0x10 , **ExpressionFailed** = FailureBit | 1 , **ExplicitFailure** = FailureBit | 2 , **Exception** = 0x100  
| FailureBit ,  
**ThrowException** = Exception | 1 , **DidntThrowException** = Exception | 2 , **FatalErrorCondition** = 0x200 |  
FailureBit }

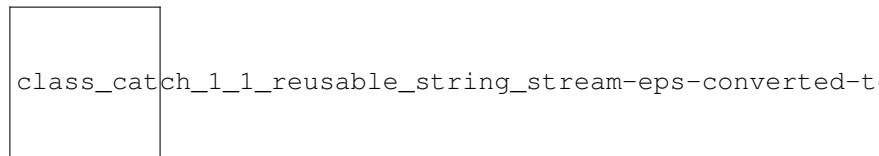
The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`



## 4.86 Catch::ReusableStringStream Class Reference

Inheritance diagram for Catch::ReusableStringStream:



### Public Member Functions

- `auto str () const` -> `std::string`
- `template<typename T >`  
`auto operator<< (T const &value) -> ReusableStringStream &`
- `auto get ()` -> `std::ostream &`

The documentation for this class was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.87 Catch::RunTests Struct Reference

### Public Types

- `enum InWhatOrder { InDeclarationOrder , InLexicographicalOrder , InRandomOrder }`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.88 Catch::ScopedMessage Class Reference

### Public Member Functions

- `ScopedMessage (MessageBuilder const &builder)`
- `ScopedMessage (ScopedMessage &duplicate)=delete`
- `ScopedMessage (ScopedMessage &&old)`

### Public Attributes

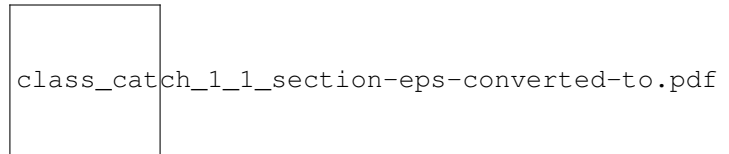
- `MessageInfo m_info`
- `bool m_moved`

The documentation for this class was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.89 Catch::Section Class Reference

Inheritance diagram for Catch::Section:



### Public Member Functions

- **Section** ([SectionInfo](#) const &info)
- **operator bool** () const

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.90 Catch::SectionEndInfo Struct Reference

### Public Attributes

- [SectionInfo](#) sectionInfo
- [Counts](#) prevAssertions
- [double](#) durationInSeconds

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.91 Catch::SectionInfo Struct Reference

### Public Member Functions

- **SectionInfo** ([SourceLineInfo](#) const &\_lineInfo, std::string const &\_name)
- **SectionInfo** ([SourceLineInfo](#) const &\_lineInfo, std::string const &\_name, std::string const &)

### Public Attributes

- std::string name
- std::string description
- [SourceLineInfo](#) lineInfo

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.92 Catch::ShowDurations Struct Reference

### Public Types

- enum **OrNot** { **DefaultForReporter** , **Always** , **Never** }

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.93 Catch::SimplePcg32 Class Reference

### Public Types

- `using result_type = std::uint32_t`

### Public Member Functions

- **SimplePcg32** (result\_type `seed_`)
- `void seed` (result\_type `seed_`)
- `void discard` (uint64\_t `skip`)
- result\_type `operator()` ()

### Static Public Member Functions

- `static constexpr result_type min` ()
- `static constexpr result_type max` ()

### Friends

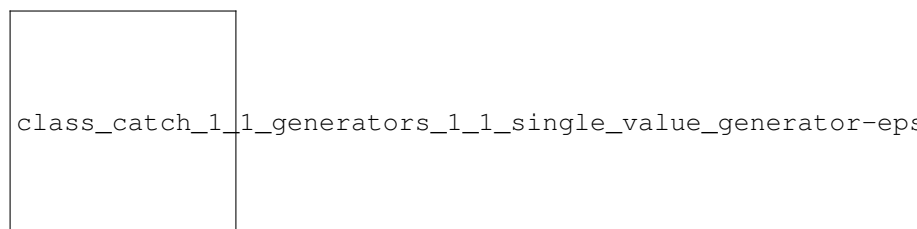
- `bool operator==` (SimplePcg32 const &lhs, SimplePcg32 const &rhs)
- `bool operator!=` (SimplePcg32 const &lhs, SimplePcg32 const &rhs)

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.94 Catch::Generators::SingleValueGenerator< T > Class Template Reference

Inheritance diagram for Catch::Generators::SingleValueGenerator< T >:



## Public Member Functions

- **SingleValueGenerator** ([T](#) &&value)
- [T](#) const & [get](#) () const override
- [bool](#) [next](#) () override

## Additional Inherited Members

## Public Types inherited from [Catch::Generators::IGenerator< T >](#)

- [using](#) type = [T](#)

### 4.94.1 Member Function Documentation

#### 4.94.1.1 [get](#)()

```
template<typename T >
T const & Catch::Generators::SingleValueGenerator< T >::get ( ) const [inline], [override],
[virtual]
```

Implements [Catch::Generators::IGenerator< T >](#).

#### 4.94.1.2 [next](#)()

```
template<typename T >
bool Catch::Generators::SingleValueGenerator< T >::next ( ) [inline], [override], [virtual]
```

Implements [Catch::Generators::GeneratorUntypedBase](#).

The documentation for this class was generated from the following file:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h](#)

## 4.95 [Catch::SourceLineInfo](#) Struct Reference

## Public Member Functions

- **SourceLineInfo** ([char](#) const \*[\\_file](#), std::size\_t [\\_line](#)) noexcept
- **SourceLineInfo** ([SourceLineInfo](#) const &[other](#))=default
- [SourceLineInfo](#) & **operator=** ([SourceLineInfo](#) const &)=default
- **SourceLineInfo** ([SourceLineInfo](#) &&) noexcept=default
- [SourceLineInfo](#) & **operator=** ([SourceLineInfo](#) &&) noexcept=default
- [bool](#) [empty](#) () const noexcept
- [bool](#) [operator==](#) ([SourceLineInfo](#) const &[other](#)) const noexcept
- [bool](#) [operator<](#) ([SourceLineInfo](#) const &[other](#)) const noexcept

**Public Attributes**

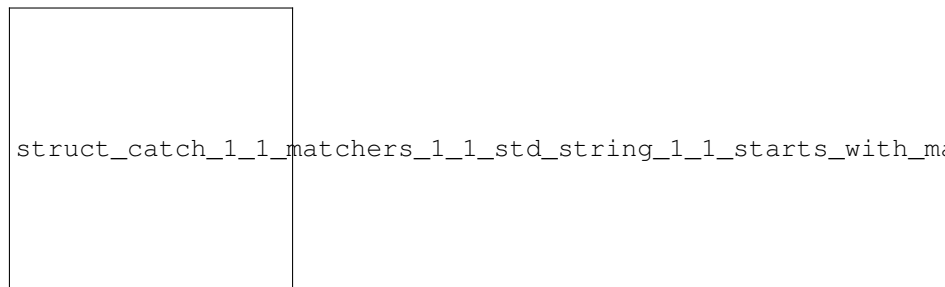
- `char const * file`
- `std::size_t line`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

**4.96 Catch::Matchers::StdString::StartsWithMatcher Struct Reference**

Inheritance diagram for Catch::Matchers::StdString::StartsWithMatcher:

**Public Member Functions**

- **StartsWithMatcher** (`CasedString const &comparator`)
- `bool match` (`std::string const &source`) `const override`

**Public Member Functions inherited from `Catch::Matchers::StdString::StringMatcherBase`**

- **StringMatcherBase** (`std::string const &operation`, `CasedString const &comparator`)
- `std::string describe` () `const override`

**Public Member Functions inherited from `Catch::Matchers::Impl::MatcherBase< T >`**

- `MatchAllOf< T > operator&&` (`MatcherBase const &other`) `const`
- `MatchAnyOf< T > operator||` (`MatcherBase const &other`) `const`
- `MatchNotOf< T > operator!` () `const`

**Public Member Functions inherited from `Catch::Matchers::Impl::MatcherUntypedBase`**

- **MatcherUntypedBase** (`MatcherUntypedBase const &`)=default
- `MatcherUntypedBase & operator=` (`MatcherUntypedBase const &`)=delete
- `std::string toString` () `const`

## Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- `virtual bool match (T const &arg) const=0`

## Additional Inherited Members

## Public Attributes inherited from [Catch::Matchers::StdString::StringMatcherBase](#)

- `CasedString m_comparator`
- `std::string m_operation`

## Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- `std::string m_cachedToString`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.97 [Catch::StreamEndStop](#) Struct Reference

### Public Member Functions

- `std::string operator+ () const`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.98 [Catch::StringMaker< T, typename >](#) Struct Template Reference

### Static Public Member Functions

- `template<typename Fake = T>  
static std::enable_if<::Catch::Detail::IsStreamInsertable< Fake >::value, std::string >::type convert (const Fake &value)`
- `template<typename Fake = T>  
static std::enable_if<!::Catch::Detail::IsStreamInsertable< Fake >::value, std::string >::type convert (const Fake &value)`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.99 Catch::StringMaker< bool > Struct Reference

### Static Public Member Functions

- [static](#) std::string **convert** ([bool b](#))

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.100 Catch::StringMaker< Catch::Detail::Approx > Struct Reference

### Static Public Member Functions

- [static](#) std::string **convert** ([Catch::Detail::Approx const](#) &value)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.101 Catch::StringMaker< char \* > Struct Reference

### Static Public Member Functions

- [static](#) std::string **convert** ([char \\*str](#))

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.102 Catch::StringMaker< char > Struct Reference

### Static Public Member Functions

- [static](#) std::string **convert** ([char c](#))

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

### 4.103 `Catch::StringMaker< char const * >` Struct Reference

#### Static Public Member Functions

- `static std::string convert (char const *str)`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

### 4.104 `Catch::StringMaker< char[SZ]>` Struct Template Reference

#### Static Public Member Functions

- `static std::string convert (char const *str)`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

### 4.105 `Catch::StringMaker< double >` Struct Reference

#### Static Public Member Functions

- `static std::string convert (double value)`

#### Static Public Attributes

- `static int precision`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

### 4.106 `Catch::StringMaker< float >` Struct Reference

#### Static Public Member Functions

- `static std::string convert (float value)`



**Static Public Attributes**

- [static int precision](#)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.107 Catch::StringMaker< int > Struct Reference

**Static Public Member Functions**

- [static std::string convert \(int value\)](#)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.108 Catch::StringMaker< long > Struct Reference

**Static Public Member Functions**

- [static std::string convert \(long value\)](#)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.109 Catch::StringMaker< long long > Struct Reference

**Static Public Member Functions**

- [static std::string convert \(long long value\)](#)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.110 `Catch::StringMaker< R C::* >` Struct Template Reference

### Static Public Member Functions

- `static std::string convert (R C::*p)`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.111 `Catch::StringMaker< R, typename std::enable_if< is_range< R >::value &&!::Catch::Detail::IsStreamInsertable< R >::value >::type >` Struct Template Reference

### Static Public Member Functions

- `static std::string convert (R const &range)`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.112 `Catch::StringMaker< signed char >` Struct Reference

### Static Public Member Functions

- `static std::string convert (signed char c)`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.113 `Catch::StringMaker< signed char[SZ]>` Struct Template Reference

### Static Public Member Functions

- `static std::string convert (signed char const *str)`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.114 Catch::StringMaker< std::nullptr\_t > Struct Reference

### Static Public Member Functions

- [static](#) std::string **convert** (std::nullptr\_t)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.115 Catch::StringMaker< std::string > Struct Reference

### Static Public Member Functions

- [static](#) std::string **convert** ([const](#) std::string &str)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.116 Catch::StringMaker< std::wstring > Struct Reference

### Static Public Member Functions

- [static](#) std::string **convert** ([const](#) std::wstring &[wstr](#))

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.117 Catch::StringMaker< T \* > Struct Template Reference

### Static Public Member Functions

- [template](#)<[typename](#) U >  
[static](#) std::string **convert** (U \*p)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.118 Catch::StringMaker< T[SZ]> Struct Template Reference

### Static Public Member Functions

- `static std::string convert (T const(&arr)[SZ])`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.119 Catch::StringMaker< unsigned char > Struct Reference

### Static Public Member Functions

- `static std::string convert (unsigned char c)`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.120 Catch::StringMaker< unsigned char[SZ]> Struct Template Reference

### Static Public Member Functions

- `static std::string convert (unsigned char const *str)`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.121 Catch::StringMaker< unsigned int > Struct Reference

### Static Public Member Functions

- `static std::string convert (unsigned int value)`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.122 Catch::StringMaker< unsigned long > Struct Reference

### Static Public Member Functions

- [static](#) std::string **convert** ([unsigned long](#) value)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.123 Catch::StringMaker< unsigned long long > Struct Reference

### Static Public Member Functions

- [static](#) std::string **convert** ([unsigned long long](#) value)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.124 Catch::StringMaker< wchar\_t \* > Struct Reference

### Static Public Member Functions

- [static](#) std::string **convert** ([wchar\\_t](#) \*str)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.125 Catch::StringMaker< wchar\_t const \* > Struct Reference

### Static Public Member Functions

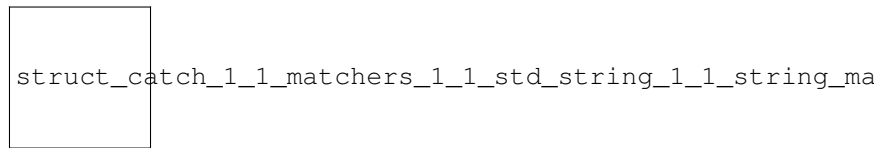
- [static](#) std::string **convert** ([wchar\\_t const](#) \*str)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.126 Catch::Matchers::StdString::StringMatcherBase Struct Reference

Inheritance diagram for Catch::Matchers::StdString::StringMatcherBase:



### Public Member Functions

- **StringMatcherBase** (std::string const &operation, CasedString const &comparator)
- std::string describe () const override

### Public Member Functions inherited from Catch::Matchers::Impl::MatcherBase< T >

- MatchAllOf< T > operator&& (MatcherBase const &other) const
- MatchAnyOf< T > operator|| (MatcherBase const &other) const
- MatchNotOf< T > operator! () const

### Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- **MatcherUntypedBase** & operator= (MatcherUntypedBase const &)=delete
- std::string toString () const

### Public Member Functions inherited from Catch::Matchers::Impl::MatcherMethod< T >

- virtual bool match (T const &arg) const=0

### Public Attributes

- CasedString m\_comparator
- std::string m\_operation

### Additional Inherited Members

### Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

- std::string m\_cachedToString

## 4.126.1 Member Function Documentation

### 4.126.1.1 describe()

```
std::string Catch::Matchers::StdString::StringMatcherBase::describe ( ) const [override],
[virtual]
```

Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.127 Catch::StringRef Class Reference

```
#include <catch.h>
```

### Public Types

- `using size_type = std::size_t`
- `using const_iterator = const char*`

### Public Member Functions

- `StringRef (char const *rawChars) noexcept`
- `constexpr StringRef (char const *rawChars, size_type size) noexcept`
- `StringRef (std::string const &stdString) noexcept`
- `operator std::string () const`
- `auto operator== (StringRef const &other) const noexcept -> bool`
- `auto operator!= (StringRef const &other) const noexcept -> bool`
- `auto operator[] (size_type index) const noexcept -> char`
- `constexpr auto empty () const noexcept -> bool`
- `constexpr auto size () const noexcept -> size_type`
- `auto c_str () const -> char const *`
- `auto substr (size_type start, size_type length) const noexcept -> StringRef`
- `auto data () const noexcept -> char const *`
- `constexpr auto isNullTerminated () const noexcept -> bool`
- `constexpr const_iterator begin () const`
- `constexpr const_iterator end () const`

### 4.127.1 Detailed Description

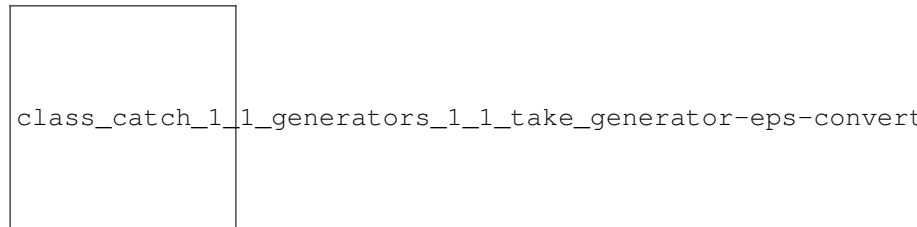
A non-owning string class (similar to the forthcoming `std::string_view`) Note that, because a [StringRef](#) may be a substring of another string, it may not be null terminated.

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.128 Catch::Generators::TakeGenerator< T > Class Template Reference

Inheritance diagram for Catch::Generators::TakeGenerator< T >:



### Public Member Functions

- **TakeGenerator** ([size\\_t](#) target, [GeneratorWrapper](#)< T > &&generator)
- [T](#) const & [get](#) () const override
- [bool](#) [next](#) () override

### Additional Inherited Members

### Public Types inherited from [Catch::Generators::IGenerator](#)< T >

- [using](#) type = T

## 4.128.1 Member Function Documentation

### 4.128.1.1 [get](#)()

```
template<typename T >
T const & Catch::Generators::TakeGenerator< T >::get ( ) const [inline], [override], [virtual]
```

Implements [Catch::Generators::IGenerator](#)< T >.

### 4.128.1.2 [next](#)()

```
template<typename T >
bool Catch::Generators::TakeGenerator< T >::next ( ) [inline], [override], [virtual]
```

Implements [Catch::Generators::GeneratorUntypedBase](#).

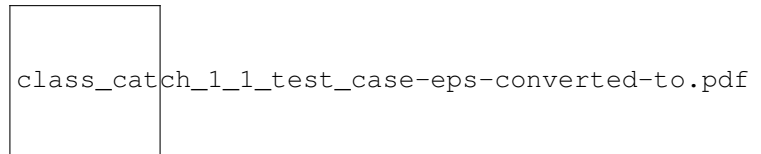
The documentation for this class was generated from the following file:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h](#)



## 4.129 Catch::TestCase Class Reference

Inheritance diagram for Catch::TestCase:



### Public Member Functions

- **TestCase** ([ITestInvoker](#) \**testCase*, [TestCaseInfo](#) &&*info*)
- **TestCase** **withName** ([std::string](#) *const* &*\_newName*) *const*
- **void** **invoke** () *const*
- [TestCaseInfo](#) *const* & **getTestCaseInfo** () *const*
- **bool** **operator==** ([TestCase](#) *const* &*other*) *const*
- **bool** **operator<** ([TestCase](#) *const* &*other*) *const*

### Public Member Functions inherited from [Catch::TestCaseInfo](#)

- **TestCaseInfo** ([std::string](#) *const* &*\_name*, [std::string](#) *const* &*\_className*, [std::string](#) *const* &*\_description*, [std::vector](#)< [std::string](#) > *const* &*\_tags*, [SourceLineInfo](#) *const* &*\_lineInfo*)
- **bool** **isHidden** () *const*
- **bool** **throws** () *const*
- **bool** **okToFail** () *const*
- **bool** **expectedToFail** () *const*
- [std::string](#) **tagsAsString** () *const*

### Additional Inherited Members

### Public Types inherited from [Catch::TestCaseInfo](#)

- enum **SpecialProperties** {  
**None** = 0 , **IsHidden** = 1 << 1 , **ShouldFail** = 1 << 2 , **MayFail** = 1 << 3 ,  
**Throws** = 1 << 4 , **NonPortable** = 1 << 5 , **Benchmark** = 1 << 6 }

### Public Attributes inherited from [Catch::TestCaseInfo](#)

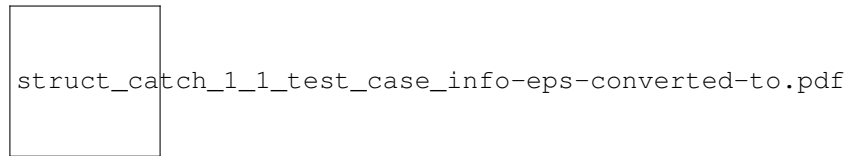
- [std::string](#) **name**
- [std::string](#) **className**
- [std::string](#) **description**
- [std::vector](#)< [std::string](#) > **tags**
- [std::vector](#)< [std::string](#) > **lcaseTags**
- [SourceLineInfo](#) **lineInfo**
- **SpecialProperties** **properties**

The documentation for this class was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.130 Catch::TestCaseInfo Struct Reference

Inheritance diagram for Catch::TestCaseInfo:



### Public Types

- enum **SpecialProperties** {  
**None** = 0 , **IsHidden** = 1 << 1 , **ShouldFail** = 1 << 2 , **MayFail** = 1 << 3 ,  
**Throws** = 1 << 4 , **NonPortable** = 1 << 5 , **Benchmark** = 1 << 6 }

### Public Member Functions

- **TestCaseInfo** (std::string [const](#) &[\\_name](#), std::string [const](#) &[\\_className](#), std::string [const](#) &[\\_description](#),  
std::vector< std::string > [const](#) &[\\_tags](#), [SourceLineInfo](#) [const](#) &[\\_lineInfo](#))
- [bool](#) **isHidden** () [const](#)
- [bool](#) **throws** () [const](#)
- [bool](#) **okToFail** () [const](#)
- [bool](#) **expectedToFail** () [const](#)
- std::string **tagsAsString** () [const](#)

### Public Attributes

- std::string **name**
- std::string **className**
- std::string **description**
- std::vector< std::string > **tags**
- std::vector< std::string > **lcaseTags**
- [SourceLineInfo](#) **lineInfo**
- SpecialProperties **properties**

### Friends

- [void](#) **setTags** ([TestCaseInfo](#) &[testCaseInfo](#), std::vector< std::string > tags)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

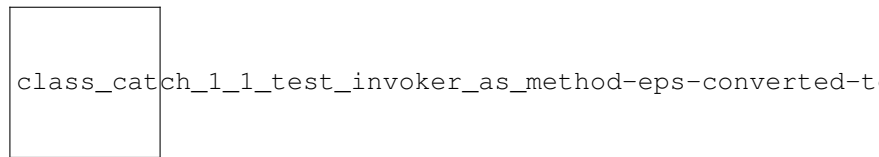
## 4.131 Catch::TestFailureException Struct Reference

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.132 Catch::TestInvokerAsMethod< C > Class Template Reference

Inheritance diagram for Catch::TestInvokerAsMethod< C >:



### Public Member Functions

- **TestInvokerAsMethod** ([void\(C::\\*testAsMethod\)\(\)](#)) [noexcept](#)
- [void invoke \(\)](#) [const override](#)

### 4.132.1 Member Function Documentation

#### 4.132.1.1 invoke()

```
template<typename C >
void Catch::TestInvokerAsMethod< C >::invoke ( ) const [inline], [override], [virtual]
```

Implements [Catch::!TestInvoker](#).

The documentation for this class was generated from the following file:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h](#)

## 4.133 Catch::Timer Class Reference

### Public Member Functions

- [void start \(\)](#)
- [auto getElapsedNanoseconds \(\)](#) [const -> uint64\\_t](#)
- [auto getElapsedMicroseconds \(\)](#) [const -> uint64\\_t](#)
- [auto getElapsedMilliseconds \(\)](#) [const -> unsigned int](#)
- [auto getElapsedSeconds \(\)](#) [const -> double](#)

The documentation for this class was generated from the following file:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h](#)

## 4.134 Catch::Totals Struct Reference

### Public Member Functions

- [Totals operator-](#) ([Totals const &other](#)) [const](#)
- [Totals & operator+=](#) ([Totals const &other](#))
- [Totals delta](#) ([Totals const &prevTotals](#)) [const](#)

**Public Attributes**

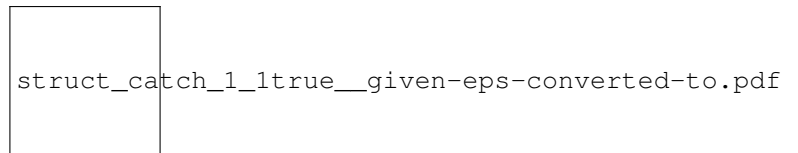
- `int error = 0`
- `Counts assertions`
- `Counts testCases`

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

**4.135 Catch::true\_given< typename > Struct Template Reference**

Inheritance diagram for Catch::true\_given< typename >:

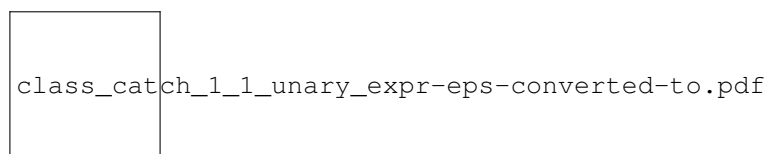


The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

**4.136 Catch::UnaryExpr< LhsT > Class Template Reference**

Inheritance diagram for Catch::UnaryExpr< LhsT >:

**Public Member Functions**

- `UnaryExpr (LhsT lhs)`

**Public Member Functions inherited from Catch::ITransientExpression**

- `auto isBinaryExpression () const -> bool`
- `auto getResult () const -> bool`
- `ITransientExpression (bool isBinaryExpression, bool result)`

**Additional Inherited Members****Public Attributes inherited from [Catch::ITransientExpression](#)**

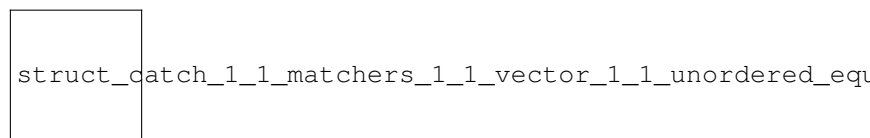
- [bool](#) `m_isBinaryExpression`
- [bool](#) `m_result`

The documentation for this class was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.137 **Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch > Struct Template Reference**

Inheritance diagram for `Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch >`:

**Public Member Functions**

- **UnorderedEqualsMatcher** (`std::vector< T, AllocComp > const &target`)
- [bool](#) **match** (`std::vector< T, AllocMatch > const &vec`) [const override](#)
- `std::string` **describe** () [const override](#)

**Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)**

- [MatchAllOf< T >](#) **operator&&** ([MatcherBase](#) `const &other`) [const](#)
- [MatchAnyOf< T >](#) **operator||** ([MatcherBase](#) `const &other`) [const](#)
- [MatchNotOf< T >](#) **operator!** () [const](#)

**Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)**

- **MatcherUntypedBase** ([MatcherUntypedBase](#) `const &`)=default
- [MatcherUntypedBase](#) & **operator=** ([MatcherUntypedBase](#) `const &`)=delete
- `std::string` **toString** () [const](#)

**Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)**

- [virtual bool](#) **match** ([T](#) `const &arg`) [const=0](#)

## Additional Inherited Members

## Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- `std::string m_cachedToString`

## 4.137.1 Member Function Documentation

### 4.137.1.1 `describe()`

```
template<typename T , typename AllocComp , typename AllocMatch >
std::string Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch >↔
::describe ( ) const [inline], [override], [virtual]
```

Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.138 [Catch::UseColour](#) Struct Reference

### Public Types

- enum `YesOrNo` { `Auto` , `Yes` , `No` }

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.139 [Var](#) Class Reference

Inheritance diagram for `Var`:



## Public Member Functions

- [Var](#) (string name)  
*Constructor for [Var](#).*
- bool [equals](#) ([Expr](#) \*e)  
*Implementation of the equals function for [Var](#).*
- int [interp](#) ()  
*the [interp\(\)](#) function for [Var](#) class.*
- bool [has\\_variable](#) ()  
*the [has\\_variable\(\)](#) function for [Var](#) class.*
- [Expr](#) \* [subst](#) (string varName, [Expr](#) \*replacement)  
*The [subst\(\)](#) function for [Var](#).*
- virtual void [print](#) (ostream &os)  
*the print function for [Var](#).*

## Public Member Functions inherited from [Expr](#)

- string [to\\_string](#) ()
- void [pretty\\_print](#) (ostream &ostream)
- virtual void [pretty\\_print\\_at](#) (ostream &ot, precedence\_t prec)
- string [to\\_pretty\\_string](#) ()

## Public Attributes

- string [name](#)

## 4.139.1 Constructor & Destructor Documentation

### 4.139.1.1 [Var\(\)](#)

```
Var::Var (
    string name )
```

Constructor for [Var](#).

#### Parameters

<i>name</i>	The integer value of the <a href="#">Num</a> object. Creates a <a href="#">Num</a> object out of val.
-------------	---

## 4.139.2 Member Function Documentation

### 4.139.2.1 [equals\(\)](#)

```
bool Var::equals (
    Expr * e ) [virtual]
```

Implementation of the equals function for [Var](#).

**Parameters**

<i>e</i>	the expression you compare.
----------	-----------------------------

**Returns**

true if name is equal, false otherwise. Verifies the current [Var](#) object is equal to a different name.

Implements [Expr](#).

**4.139.2.2 has\_variable()**

```
bool Var::has_variable ( ) [virtual]
```

the [has\\_variable\(\)](#) function for [Var](#) class.

**Returns**

ALWAYS will return true. Verifies that a variable is a variable.

Implements [Expr](#).

**4.139.2.3 interp()**

```
int Var::interp ( ) [virtual]
```

the [interp\(\)](#) function for [Var](#) class.

**Returns**

runtime error.

Implements [Expr](#).

**4.139.2.4 print()**

```
void Var::print (
    ostream & ostream ) [virtual]
```

the print function for [Var](#).

**Parameters**

<i>ostream</i>	The output stream to print to. Prints the value of the <a href="#">Var</a> object to the specified output stream.
----------------	---

Implements [Expr](#).



## 4.139.2.5 subst()

```
Expr * Var::subst (
    string varName,
    Expr * replacement ) [virtual]
```

The `subst()` function for `Var`.

## Parameters

<i>varName</i>	the variable that will be replaced.
<i>replacement</i>	The replacement expression.

## Returns

the replacement, or a new `Var` instance with the same name.

Implements `Expr`.

The documentation for this class was generated from the following files:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.h](#)
- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.cpp](#)

## 4.140 Catch::detail::void\_type&lt;... &gt; Struct Template Reference

## Public Types

- `using type = void`

The documentation for this struct was generated from the following file:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h](#)

## 4.141 Catch::WaitForKeypress Struct Reference

## Public Types

- enum `When` { `Never` , `BeforeStart` = 1 , `BeforeExit` = 2 , `BeforeStartAndExit` = `BeforeStart` | `BeforeExit` }

The documentation for this struct was generated from the following file:

- [/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h](#)

## 4.142 Catch::WarnAbout Struct Reference

### Public Types

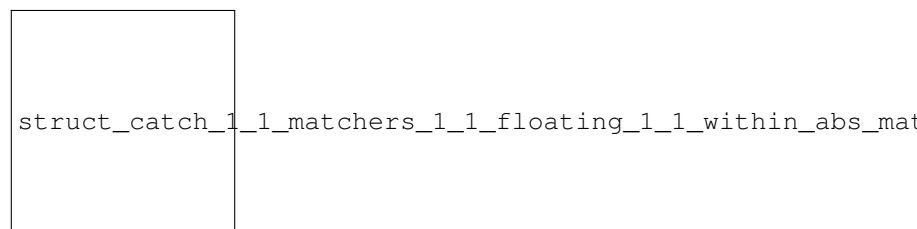
- enum **What** { **Nothing** = 0x00 , **NoAssertions** = 0x01 , **NoTests** = 0x02 }

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

## 4.143 Catch::Matchers::Floating::WithinAbsMatcher Struct Reference

Inheritance diagram for Catch::Matchers::Floating::WithinAbsMatcher:



### Public Member Functions

- WithinAbsMatcher** (double target, double margin)
- bool match** (double const &matchee) const override
- std::string **describe** () const override

### Public Member Functions inherited from **Catch::Matchers::Impl::MatcherBase< T >**

- MatchAllOf**< T > **operator&&** (MatcherBase const &other) const
- MatchAnyOf**< T > **operator||** (MatcherBase const &other) const
- MatchNotOf**< T > **operator!** () const

### Public Member Functions inherited from **Catch::Matchers::Impl::MatcherUntypedBase**

- MatcherUntypedBase** (MatcherUntypedBase const &)=default
- MatcherUntypedBase** & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

### Public Member Functions inherited from **Catch::Matchers::Impl::MatcherMethod< T >**

- virtual bool match** (T const &arg) const=0

## Additional Inherited Members

### Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- `std::string m_cachedToString`

## 4.143.1 Member Function Documentation

### 4.143.1.1 `describe()`

```
std::string Catch::Matchers::Floating::WithinAbsMatcher::describe ( ) const [override], [virtual]
```

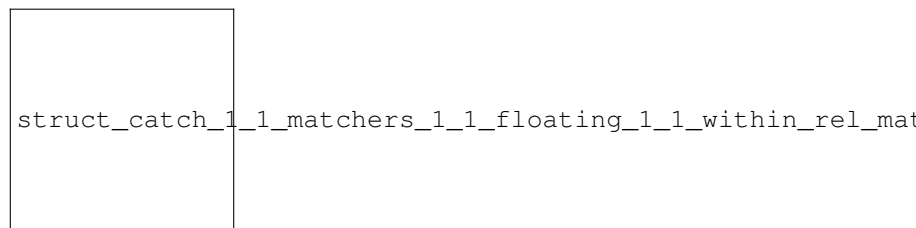
Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

## 4.144 Catch::Matchers::Floating::WithinRelMatcher Struct Reference

Inheritance diagram for `Catch::Matchers::Floating::WithinRelMatcher`:



## Public Member Functions

- **WithinRelMatcher** (`double target`, `double epsilon`)
- **bool match** (`double const &matchee`) `const override`
- `std::string describe () const override`

## Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)

- **MatchAllOf**< `T` > **operator&&** (`MatcherBase const &other`) `const`
- **MatchAnyOf**< `T` > **operator||** (`MatcherBase const &other`) `const`
- **MatchNotOf**< `T` > **operator!** () `const`

## Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- **MatcherUntypedBase** (`MatcherUntypedBase const &`)=`default`
- **MatcherUntypedBase & operator=** (`MatcherUntypedBase const &`)=`delete`
- `std::string toString () const`

## Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- [virtual bool match](#) ([T const](#) &arg) [const=0](#)

## Additional Inherited Members

## Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- [std::string m\\_cachedToString](#)

### 4.144.1 Member Function Documentation

#### 4.144.1.1 [describe\(\)](#)

`std::string Catch::Matchers::Floating::WithinRelMatcher::describe ( ) const [override], [virtual]`

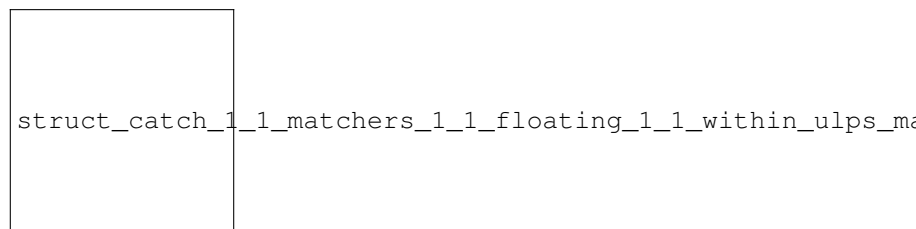
Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

The documentation for this struct was generated from the following file:

- `/Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h`

### 4.145 [Catch::Matchers::Floating::WithinUlpMatcher](#) Struct Reference

Inheritance diagram for [Catch::Matchers::Floating::WithinUlpMatcher](#):



## Public Member Functions

- [WithinUlpMatcher](#) ([double target](#), [uint64\\_t ulps](#), [FloatingPointKind baseType](#))
- [bool match](#) ([double const](#) &matchee) [const override](#)
- [std::string describe](#) () [const override](#)

## Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)

- [MatchAllOf< T > operator&&](#) ([MatcherBase const](#) &other) [const](#)
- [MatchAnyOf< T > operator||](#) ([MatcherBase const](#) &other) [const](#)
- [MatchNotOf< T > operator!](#) () [const](#)

**Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)**

- **MatcherUntypedBase** ([MatcherUntypedBase const &](#))=default
- [MatcherUntypedBase &](#) **operator=** ([MatcherUntypedBase const &](#))=delete
- std::string **toString** () [const](#)

**Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)**

- [virtual bool](#) **match** ([T const &arg](#)) [const=0](#)

**Additional Inherited Members****Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)**

- std::string **m\_cachedToString**

**4.145.1 Member Function Documentation****4.145.1.1 describe()**

```
std::string Catch::Matchers::Floating::WithinUlpMatcher::describe ( ) const [override],  
[virtual]
```

Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h



## Chapter 5

# File Documentation

### 5.1 /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

```
00001 /*
00002  * Catch v2.13.10
00003  * Generated: 2022-10-16 11:01:23.452308
00004  * -----
00005  * This file has been merged from multiple headers. Please don't edit it directly
00006  * Copyright (c) 2022 Two Blue Cubes Ltd. All rights reserved.
00007  *
00008  * Distributed under the Boost Software License, Version 1.0. (See accompanying
00009  * file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
00010  */
00011 #ifndef TWOBLUECUBES_SINGLE_INCLUDE_CATCH_HPP_INCLUDED
00012 #define TWOBLUECUBES_SINGLE_INCLUDE_CATCH_HPP_INCLUDED
00013 // start catch.hpp
00014
00015
00016 #define CATCH_VERSION_MAJOR 2
00017 #define CATCH_VERSION_MINOR 13
00018 #define CATCH_VERSION_PATCH 10
00019
00020 #ifdef __clang__
00021 # pragma clang system_header
00022 #elif defined __GNUC__
00023 # pragma GCC system_header
00024 #endif
00025
00026 // start catch_suppress_warnings.h
00027
00028 #ifdef __clang__
00029 # ifdef __ICC // icpc defines the __clang__ macro
00030 # pragma warning(push)
00031 # pragma warning(disable: 161 1682)
00032 # else // __ICC
00033 # pragma clang diagnostic push
00034 # pragma clang diagnostic ignored "-Wpadded"
00035 # pragma clang diagnostic ignored "-Wswitch-enum"
00036 # pragma clang diagnostic ignored "-Wcovered-switch-default"
00037 # endif
00038 #elif defined __GNUC__
00039 // Because REQUIREs trigger GCC's -Wparentheses, and because still
00040 // supported version of g++ have only buggy support for _Pragmas,
00041 // Wparentheses have to be suppressed globally.
00042 # pragma GCC diagnostic ignored "-Wparentheses" // See #674 for details
00043
00044 # pragma GCC diagnostic push
00045 # pragma GCC diagnostic ignored "-Wunused-variable"
00046 # pragma GCC diagnostic ignored "-Wpadded"
00047 #endif
00048 // end catch_suppress_warnings.h
00049 #if defined(CATCH_CONFIG_MAIN) || defined(CATCH_CONFIG_RUNNER)
00050 # define CATCH_IMPL
00051 # define CATCH_CONFIG_ALL_PARTS
00052 #endif
00053
00054 // In the impl file, we want to have access to all parts of the headers
00055 // Can also be used to sanely support PCHs
00056 #if defined(CATCH_CONFIG_ALL_PARTS)
00057 # define CATCH_CONFIG_EXTERNAL_INTERFACES
00058 # if defined(CATCH_CONFIG_DISABLE_MATCHERS)
```

```

00059 #    undef CATCH_CONFIG_DISABLE_MATCHERS
00060 #    endif
00061 #    if !defined(CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER)
00062 #        define CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER
00063 #    endif
00064 #endif
00065
00066 #if !defined(CATCH_CONFIG_IMPL_ONLY)
00067 // start catch_platform.h
00068
00069 // See e.g.:
00070 // https://opensource.apple.com/source/CarbonHeaders/CarbonHeaders-18.1/TargetConditionals.h.auto.html
00071 #ifdef __APPLE__
00072 #    include <TargetConditionals.h>
00073 #    if (defined(TARGET_OS_OSX) && TARGET_OS_OSX == 1) || \
00074         (defined(TARGET_OS_MAC) && TARGET_OS_MAC == 1)
00075 #        define CATCH_PLATFORM_MAC
00076 #    elif (defined(TARGET_OS_IPHONE) && TARGET_OS_IPHONE == 1)
00077 #        define CATCH_PLATFORM_IPHONE
00078 #    endif
00079
00080 #elif defined(linux) || defined(__linux) || defined(__linux__)
00081 #    define CATCH_PLATFORM_LINUX
00082
00083 #elif defined(WIN32) || defined(__WIN32__) || defined(_WIN32) || defined(_MSC_VER) || \
00084         defined(__MINGW32__)
00085 #    define CATCH_PLATFORM_WINDOWS
00086 #endif
00087 // end catch_platform.h
00088
00089 #ifdef CATCH_IMPL
00090 #    ifndef CLARA_CONFIG_MAIN
00091 #        define CLARA_CONFIG_MAIN_NOT_DEFINED
00092 #        define CLARA_CONFIG_MAIN
00093 #    endif
00094 #endif
00095
00096 // start catch_user_interfaces.h
00097
00098 namespace Catch {
00099     unsigned int rngSeed();
00100 }
00101
00102 // end catch_user_interfaces.h
00103 // start catch_tag_alias_autoregistrar.h
00104
00105 // start catch_common.h
00106
00107 // start catch_compiler_capabilities.h
00108
00109 // Detect a number of compiler features - by compiler
00110 // The following features are defined:
00111 //
00112 // CATCH_CONFIG_COUNTER : is the __COUNTER__ macro supported?
00113 // CATCH_CONFIG_WINDOWS_SEH : is Windows SEH supported?
00114 // CATCH_CONFIG_POSIX_SIGNALS : are POSIX signals supported?
00115 // CATCH_CONFIG_DISABLE_EXCEPTIONS : Are exceptions enabled?
00116 // *****
00117 // Note to maintainers: if new toggles are added please document them
00118 // in configuration.md, too
00119 // *****
00120
00121 // In general each macro has a _NO_<feature name> form
00122 // (e.g. CATCH_CONFIG_NO_POSIX_SIGNALS) which disables the feature.
00123 // Many features, at point of detection, define an _INTERNAL_ macro, so they
00124 // can be combined, en-mass, with the _NO_ forms later.
00125
00126 #ifdef __cplusplus
00127
00128 #    if (__cplusplus >= 201402L) || (defined(_MSVC_LANG) && _MSVC_LANG >= 201402L)
00129 #        define CATCH_CPP14_OR_GREATER
00130 #    endif
00131
00132 #    if (__cplusplus >= 201703L) || (defined(_MSVC_LANG) && _MSVC_LANG >= 201703L)
00133 #        define CATCH_CPP17_OR_GREATER
00134 #    endif
00135 #endif
00136 #endif
00137
00138 // Only GCC compiler should be used in this block, so other compilers trying to
00139 // mask themselves as GCC should be ignored.
00140 #if defined(__GNUC__) && !defined(__clang__) && !defined(__ICC) && !defined(__CUDACC__) && \
    !defined(__LCC__)
00141 #    define CATCH_INTERNAL_START_WARNINGS_SUPPRESSION _Pragma( "GCC diagnostic push" )
00142 #    define CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION _Pragma( "GCC diagnostic pop" )
00143

```



```

00144 #   define CATCH_INTERNAL_IGNORE_BUT_WARN(...) (void)__builtin_constant_p(__VA_ARGS__)
00145
00146 #endif
00147
00148 #if defined(__clang__)
00149
00150 #   define CATCH_INTERNAL_START_WARNINGS_SUPPRESSION _Pragma( "clang diagnostic push" )
00151 #   define CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION  _Pragma( "clang diagnostic pop" )
00152
00153 // As of this writing, IBM XL's implementation of __builtin_constant_p has a bug
00154 // which results in calls to destructors being emitted for each temporary,
00155 // without a matching initialization. In practice, this can result in something
00156 // like `std::string::~string` being called on an uninitialized value.
00157 //
00158 // For example, this code will likely segfault under IBM XL:
00159 // ```
00160 // REQUIRE(std::string("12") + "34" == "1234")
00161 // ```
00162 //
00163 // Therefore, `CATCH_INTERNAL_IGNORE_BUT_WARN` is not implemented.
00164 #   if !defined(__ibmxl__) && !defined(__CUDACC__)
00165 #       define CATCH_INTERNAL_IGNORE_BUT_WARN(...) (void)__builtin_constant_p(__VA_ARGS__) /*
00166         NOLINT(cppcoreguidelines-pro-type-vararg, hicpp-vararg) */
00167 #   endif
00168
00169 #   define CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
00170     _Pragma( "clang diagnostic ignored \\"-Wexit-time-destructors\\" ) \
00171     _Pragma( "clang diagnostic ignored \\"-Wglobal-constructors\\" )
00172
00173 #   define CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS \
00174     _Pragma( "clang diagnostic ignored \\"-Wparentheses\\" )
00175
00176 #   define CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS \
00177     _Pragma( "clang diagnostic ignored \\"-Wunused-variable\\" )
00178
00179 #   define CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
00180     _Pragma( "clang diagnostic ignored \\"-Wgnu-zero-variadic-macro-arguments\\" )
00181
00182 #   define CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
00183     _Pragma( "clang diagnostic ignored \\"-Wunused-template\\" )
00184 #endif // __clang__
00185
00186 // Assume that non-Windows platforms support posix signals by default
00187 #if !defined(CATCH_PLATFORM_WINDOWS)
00188 #define CATCH_INTERNAL_CONFIG_POSIX_SIGNALS
00189 #endif
00190
00191 // We know some environments not to support full POSIX signals
00192 #if defined(__CYGWIN__) || defined(__QNX__) || defined(__EMSCRIPTEN__) || defined(__DJGPP__)
00193 #define CATCH_INTERNAL_CONFIG_NO_POSIX_SIGNALS
00194 #endif
00195
00196 #ifdef __OS400__
00197 #define CATCH_INTERNAL_CONFIG_NO_POSIX_SIGNALS
00198 #define CATCH_CONFIG_COLOUR_NONE
00199 #endif
00200
00201 // Android somehow still does not support std::to_string
00202 #if defined(__ANDROID__)
00203 #define CATCH_INTERNAL_CONFIG_NO_CPP11_TO_STRING
00204 #define CATCH_INTERNAL_CONFIG_ANDROID_LOGWRITE
00205 #endif
00206
00207 // Not all Windows environments support SEH properly
00208 #if defined(__MINGW32__)
00209 #define CATCH_INTERNAL_CONFIG_NO_WINDOWS_SEH
00210 #endif
00211
00212 // PS4
00213 #if defined(__ORBIS__)
00214 #define CATCH_INTERNAL_CONFIG_NO_NEW_CAPTURE
00215 #endif
00216
00217 // Cygwin
00218 #ifdef __CYGWIN__
00219 // Required for some versions of Cygwin to declare gettimeofday
00220 // see: http://stackoverflow.com/questions/36901803/gettimeofday-not-declared-in-this-scope-cygwin
00221 #define _BSD_SOURCE
00222 // some versions of cygwin (most) do not support std::to_string. Use the libstd check.
00223 // https://gcc.gnu.org/onlinedocs/gcc-4.8.2/libstdc++/api/a01053_source.html line 2812-2813
00224 #if !((__cplusplus >= 201103L) && defined(_GLIBCXX_USE_C99) \
00225     && !defined(_GLIBCXX_HAVE_BROKEN_VSWPRINTF))
00226 #define CATCH_INTERNAL_CONFIG_NO_CPP11_TO_STRING
00227 #endif
00228
00229 #endif

```

```

00236 # endif
00237 #endif // __CYGWIN__
00238
00240 // Visual C++
00241 #if defined(_MSC_VER)
00242
00243 // Universal Windows platform does not support SEH
00244 // Or console colours (or console at all...)
00245 # if defined(WINAPI_FAMILY) && (WINAPI_FAMILY == WINAPI_FAMILY_APP)
00246 #   define CATCH_CONFIG_COLOUR_NONE
00247 # else
00248 #   define CATCH_INTERNAL_CONFIG_WINDOWS_SEH
00249 # endif
00250
00251 # if !defined(__clang__) // Handle Clang masquerading for msvc
00252
00253 // MSVC traditional preprocessor needs some workaround for __VA_ARGS__
00254 // _MSVC_TRADITIONAL == 0 means new conformant preprocessor
00255 // _MSVC_TRADITIONAL == 1 means old traditional non-conformant preprocessor
00256 #   if !defined(_MSVC_TRADITIONAL) || (defined(_MSVC_TRADITIONAL) && _MSVC_TRADITIONAL)
00257 #       define CATCH_INTERNAL_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
00258 #   endif // _MSVC_TRADITIONAL
00259
00260 // Only do this if we're not using clang on Windows, which uses `diagnostic push` & `diagnostic pop`
00261 #   define CATCH_INTERNAL_START_WARNINGS_SUPPRESSION __pragma( warning(push) )
00262 #   define CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION __pragma( warning(pop) )
00263 #   endif // __clang__
00264
00265 #endif // _MSC_VER
00266
00267 #if defined(_REENTRANT) || defined(_MSC_VER)
00268 // Enable async processing, as -pthread is specified or no additional linking is required
00269 # define CATCH_INTERNAL_CONFIG_USE_ASYNC
00270 #endif // _MSC_VER
00271
00273 // Check if we are compiled with -fno-exceptions or equivalent
00274 #if defined(__EXCEPTIONS) || defined(__cpp_exceptions) || defined(_CPPUNWIND)
00275 #   define CATCH_INTERNAL_CONFIG_EXCEPTIONS_ENABLED
00276 #endif
00277
00279 // DJGPP
00280 #ifdef __DJGPP__
00281 #   define CATCH_INTERNAL_CONFIG_NO_WCHAR
00282 #endif // __DJGPP__
00283
00285 // Embarcadero C++Build
00286 #if defined(__BORLANDC__)
00287 #   define CATCH_INTERNAL_CONFIG_POLYFILL_ISNAN
00288 #endif
00289
00291
00292 // Use of __COUNTER__ is suppressed during code analysis in
00293 // CLion/AppCode 2017.2.x and former, because __COUNTER__ is not properly
00294 // handled by it.
00295 // Otherwise all supported compilers support COUNTER macro,
00296 // but user still might want to turn it off
00297 #if ( !defined(__JETBRAINS_IDE__) || __JETBRAINS_IDE__ >= 20170300L )
00298 #   define CATCH_INTERNAL_CONFIG_COUNTER
00299 #endif
00300
00302
00303 // RTX is a special version of Windows that is real time.
00304 // This means that it is detected as Windows, but does not provide
00305 // the same set of capabilities as real Windows does.
00306 #if defined(UNDER_RTSS) || defined(RTX64_BUILD)
00307 #   define CATCH_INTERNAL_CONFIG_NO_WINDOWS_SEH
00308 #   define CATCH_INTERNAL_CONFIG_NO_ASYNC
00309 #   define CATCH_CONFIG_COLOUR_NONE
00310 #endif
00311
00312 #if !defined(_GLIBCXX_USE_C99_MATH_TR1)
00313 #   define CATCH_INTERNAL_CONFIG_GLOBAL_NEXTAFTER
00314 #endif
00315
00316 // Various stdlib support checks that require __has_include
00317 #if defined(__has_include)
00318 // Check if string_view is available and usable
00319 #   if __has_include(<string_view>) && defined(CATCH_CPP17_OR_GREATER)
00320 #       define CATCH_INTERNAL_CONFIG_CPP17_STRING_VIEW
00321 #   endif
00322
00323 // Check if optional is available and usable
00324 #   if __has_include(<optional>) && defined(CATCH_CPP17_OR_GREATER)
00325 #       define CATCH_INTERNAL_CONFIG_CPP17_OPTIONAL
00326 #   endif // __has_include(<optional>) && defined(CATCH_CPP17_OR_GREATER)
00327
00328 // Check if byte is available and usable

```

```

00329 # if __has_include(<cstdint>) && defined(CATCH_CPP17_OR_GREATER)
00330 #     include <cstdint>
00331 #     if defined(__cpp_lib_byte) && (__cpp_lib_byte > 0)
00332 #         define CATCH_INTERNAL_CONFIG_CPP17_BYTE
00333 #     endif
00334 # endif // __has_include(<cstdint>) && defined(CATCH_CPP17_OR_GREATER)
00335
00336 // Check if variant is available and usable
00337 # if __has_include(<variant>) && defined(CATCH_CPP17_OR_GREATER)
00338 #     if defined(__clang__) && (__clang_major__ < 8)
00339         // work around clang bug with libstdc++ https://bugs.llvm.org/show_bug.cgi?id=31852
00340         // fix should be in clang 8, workaround in libstdc++ 8.2
00341 #         include <ciso646>
00342 #         if defined(__GLIBCXX__) && defined(_GLIBCXX_RELEASE) && (_GLIBCXX_RELEASE < 9)
00343 #             define CATCH_CONFIG_NO_CPP17_VARIANT
00344 #         else
00345 #             define CATCH_INTERNAL_CONFIG_CPP17_VARIANT
00346 #         endif // defined(__GLIBCXX__) && defined(_GLIBCXX_RELEASE) && (_GLIBCXX_RELEASE < 9)
00347 #     else
00348 #         define CATCH_INTERNAL_CONFIG_CPP17_VARIANT
00349 #     endif // defined(__clang__) && (__clang_major__ < 8)
00350 # endif // __has_include(<variant>) && defined(CATCH_CPP17_OR_GREATER)
00351 #endif // defined(__has_include)
00352
00353 #if defined(CATCH_INTERNAL_CONFIG_COUNTER) && !defined(CATCH_CONFIG_NO_COUNTER) &&
    !defined(CATCH_CONFIG_COUNTER)
00354 #     define CATCH_CONFIG_COUNTER
00355 #endif
00356 #if defined(CATCH_INTERNAL_CONFIG_WINDOWS_SEH) && !defined(CATCH_CONFIG_NO_WINDOWS_SEH) &&
    !defined(CATCH_CONFIG_WINDOWS_SEH) && !defined(CATCH_INTERNAL_CONFIG_NO_WINDOWS_SEH)
00357 #     define CATCH_CONFIG_WINDOWS_SEH
00358 #endif
00359 // This is set by default, because we assume that unix compilers are posix-signal-compatible by
    default.
00360 #if defined(CATCH_INTERNAL_CONFIG_POSIX_SIGNALS) && !defined(CATCH_INTERNAL_CONFIG_NO_POSIX_SIGNALS)
    && !defined(CATCH_CONFIG_NO_POSIX_SIGNALS) && !defined(CATCH_CONFIG_POSIX_SIGNALS)
00361 #     define CATCH_CONFIG_POSIX_SIGNALS
00362 #endif
00363 // This is set by default, because we assume that compilers with no wchar_t support are just rare
    exceptions.
00364 #if !defined(CATCH_INTERNAL_CONFIG_NO_WCHAR) && !defined(CATCH_CONFIG_NO_WCHAR) &&
    !defined(CATCH_CONFIG_WCHAR)
00365 #     define CATCH_CONFIG_WCHAR
00366 #endif
00367
00368 #if !defined(CATCH_INTERNAL_CONFIG_NO_CPP11_TO_STRING) && !defined(CATCH_CONFIG_NO_CPP11_TO_STRING) &&
    !defined(CATCH_CONFIG_CPP11_TO_STRING)
00369 #     define CATCH_CONFIG_CPP11_TO_STRING
00370 #endif
00371
00372 #if defined(CATCH_INTERNAL_CONFIG_CPP17_OPTIONAL) && !defined(CATCH_CONFIG_NO_CPP17_OPTIONAL) &&
    !defined(CATCH_CONFIG_CPP17_OPTIONAL)
00373 #     define CATCH_CONFIG_CPP17_OPTIONAL
00374 #endif
00375
00376 #if defined(CATCH_INTERNAL_CONFIG_CPP17_STRING_VIEW) && !defined(CATCH_CONFIG_NO_CPP17_STRING_VIEW) &&
    !defined(CATCH_CONFIG_CPP17_STRING_VIEW)
00377 #     define CATCH_CONFIG_CPP17_STRING_VIEW
00378 #endif
00379
00380 #if defined(CATCH_INTERNAL_CONFIG_CPP17_VARIANT) && !defined(CATCH_CONFIG_NO_CPP17_VARIANT) &&
    !defined(CATCH_CONFIG_CPP17_VARIANT)
00381 #     define CATCH_CONFIG_CPP17_VARIANT
00382 #endif
00383
00384 #if defined(CATCH_INTERNAL_CONFIG_CPP17_BYTE) && !defined(CATCH_CONFIG_NO_CPP17_BYTE) &&
    !defined(CATCH_CONFIG_CPP17_BYTE)
00385 #     define CATCH_CONFIG_CPP17_BYTE
00386 #endif
00387
00388 #if defined(CATCH_CONFIG_EXPERIMENTAL_REDIRECT)
00389 #     define CATCH_INTERNAL_CONFIG_NEW_CAPTURE
00390 #endif
00391
00392 #if defined(CATCH_INTERNAL_CONFIG_NEW_CAPTURE) && !defined(CATCH_INTERNAL_CONFIG_NO_NEW_CAPTURE) &&
    !defined(CATCH_CONFIG_NO_NEW_CAPTURE) && !defined(CATCH_CONFIG_NEW_CAPTURE)
00393 #     define CATCH_CONFIG_NEW_CAPTURE
00394 #endif
00395
00396 #if !defined(CATCH_INTERNAL_CONFIG_EXCEPTIONS_ENABLED) && !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
00397 #     define CATCH_CONFIG_DISABLE_EXCEPTIONS
00398 #endif
00399
00400 #if defined(CATCH_INTERNAL_CONFIG_POLYFILL_ISNAN) && !defined(CATCH_CONFIG_NO_POLYFILL_ISNAN) &&
    !defined(CATCH_CONFIG_POLYFILL_ISNAN)
00401 #     define CATCH_CONFIG_POLYFILL_ISNAN
00402 #endif

```

```

00403
00404 #if defined(CATCH_INTERNAL_CONFIG_USE_ASYNC) && !defined(CATCH_INTERNAL_CONFIG_NO_ASYNC) &&
!defined(CATCH_CONFIG_NO_USE_ASYNC) && !defined(CATCH_CONFIG_USE_ASYNC)
00405 # define CATCH_CONFIG_USE_ASYNC
00406 #endif
00407
00408 #if defined(CATCH_INTERNAL_CONFIG_ANDROID_LOGWRITE) && !defined(CATCH_CONFIG_NO_ANDROID_LOGWRITE) &&
!defined(CATCH_CONFIG_ANDROID_LOGWRITE)
00409 # define CATCH_CONFIG_ANDROID_LOGWRITE
00410 #endif
00411
00412 #if defined(CATCH_INTERNAL_CONFIG_GLOBAL_NEXTAFTER) && !defined(CATCH_CONFIG_NO_GLOBAL_NEXTAFTER) &&
!defined(CATCH_CONFIG_GLOBAL_NEXTAFTER)
00413 # define CATCH_CONFIG_GLOBAL_NEXTAFTER
00414 #endif
00415
00416 // Even if we do not think the compiler has that warning, we still have
00417 // to provide a macro that can be used by the code.
00418 #if !defined(CATCH_INTERNAL_START_WARNINGS_SUPPRESSION)
00419 # define CATCH_INTERNAL_START_WARNINGS_SUPPRESSION
00420 #endif
00421 #if !defined(CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION)
00422 # define CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
00423 #endif
00424 #if !defined(CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS)
00425 # define CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS
00426 #endif
00427 #if !defined(CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS)
00428 # define CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS
00429 #endif
00430 #if !defined(CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS)
00431 # define CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS
00432 #endif
00433 #if !defined(CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS)
00434 # define CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS
00435 #endif
00436
00437 // The goal of this macro is to avoid evaluation of the arguments, but
00438 // still have the compiler warn on problems inside...
00439 #if !defined(CATCH_INTERNAL_IGNORE_BUT_WARN)
00440 # define CATCH_INTERNAL_IGNORE_BUT_WARN(...)
00441 #endif
00442
00443 #if defined(__APPLE__) && defined(__apple_build_version__) && (__clang_major__ < 10)
00444 # undef CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS
00445 #elif defined(__clang__) && (__clang_major__ < 5)
00446 # undef CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS
00447 #endif
00448
00449 #if !defined(CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS)
00450 # define CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS
00451 #endif
00452
00453 #if defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
00454 #define CATCH_TRY if ((true))
00455 #define CATCH_CATCH_ALL if ((false))
00456 #define CATCH_CATCH_ANON(type) if ((false))
00457 #else
00458 #define CATCH_TRY try
00459 #define CATCH_CATCH_ALL catch (...)
00460 #define CATCH_CATCH_ANON(type) catch (type)
00461 #endif
00462
00463 #if defined(CATCH_INTERNAL_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR) &&
!defined(CATCH_CONFIG_NO_TRADITIONAL_MSVC_PREPROCESSOR) &&
!defined(CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR)
00464 #define CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
00465 #endif
00466
00467 // end catch_compiler_capabilities.h
00468 #define INTERNAL_CATCH_UNIQUE_NAME_LINE2( name, line ) name##line
00469 #define INTERNAL_CATCH_UNIQUE_NAME_LINE( name, line ) INTERNAL_CATCH_UNIQUE_NAME_LINE2( name, line )
00470 #ifdef CATCH_CONFIG_COUNTER
00471 # define INTERNAL_CATCH_UNIQUE_NAME( name ) INTERNAL_CATCH_UNIQUE_NAME_LINE( name, __COUNTER__ )
00472 #else
00473 # define INTERNAL_CATCH_UNIQUE_NAME( name ) INTERNAL_CATCH_UNIQUE_NAME_LINE( name, __LINE__ )
00474 #endif
00475
00476 #include <iosfwd>
00477 #include <string>
00478 #include <stdint>
00479
00480 // We need a dummy global operator« so we can bring it into Catch namespace later
00481 struct Catch_global_namespace_dummy {};
00482 std::ostream& operator«(std::ostream&, Catch_global_namespace_dummy);
00483
00484 namespace Catch {

```

```

00485
00486     struct CaseSensitive { enum Choice {
00487         Yes,
00488         No
00489     }; };
00490
00491     class NonCopyable {
00492     public:
00493         NonCopyable( NonCopyable const& )           = delete;
00494         NonCopyable( NonCopyable && )               = delete;
00495         NonCopyable& operator = ( NonCopyable const& ) = delete;
00496         NonCopyable& operator = ( NonCopyable && )   = delete;
00497
00498     protected:
00499         NonCopyable();
00500         virtual ~NonCopyable();
00501     };
00502
00503     struct SourceLineInfo {
00504     public:
00505         SourceLineInfo() = delete;
00506         SourceLineInfo( char const* _file, std::size_t _line ) noexcept
00507             : file( _file ),
00508               line( _line )
00509         {}
00510
00511         SourceLineInfo( SourceLineInfo const& other )           = default;
00512         SourceLineInfo& operator = ( SourceLineInfo const& )   = default;
00513         SourceLineInfo( SourceLineInfo&& )                     noexcept = default;
00514         SourceLineInfo& operator = ( SourceLineInfo&& )         noexcept = default;
00515
00516         bool empty() const noexcept { return file[0] == '\0'; }
00517         bool operator == ( SourceLineInfo const& other ) const noexcept;
00518         bool operator < ( SourceLineInfo const& other ) const noexcept;
00519
00520         char const* file;
00521         std::size_t line;
00522     };
00523
00524     std::ostream& operator << ( std::ostream& os, SourceLineInfo const& info );
00525
00526     // Bring in operator<< from global namespace into Catch namespace
00527     // This is necessary because the overload of operator<< above makes
00528     // lookup stop at namespace Catch
00529     using ::operator<<;
00530
00531     // Use this in variadic streaming macros to allow
00532     // » +StreamEndStop
00533     // as well as
00534     // » stuff +StreamEndStop
00535     struct StreamEndStop {
00536     public:
00537         std::string operator+() const;
00538     };
00539     template<typename T>
00540     T const& operator + ( T const& value, StreamEndStop ) {
00541     return value;
00542     }
00543
00544 #define CATCH_INTERNAL_LINEINFO \
00545     ::Catch::SourceLineInfo( __FILE__, static_cast<std::size_t>( __LINE__ ) )
00546
00547 // end catch_common.h
00548 namespace Catch {
00549
00550     struct RegistrarForTagAliases {
00551     public:
00552         RegistrarForTagAliases( char const* alias, char const* tag, SourceLineInfo const& lineInfo );
00553     };
00554
00555 } // end namespace Catch
00556
00557 #define CATCH_REGISTER_TAG_ALIAS( alias, spec ) \
00558     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
00559     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
00560     namespace{ Catch::RegistrarForTagAliases INTERNAL_CATCH_UNIQUE_NAME( AutoRegisterTagAlias )( \
00561         alias, spec, CATCH_INTERNAL_LINEINFO ); } \
00562     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
00563
00564 // end catch_tag_alias_autoregistrar.h
00565 // start catch_test_registry.h
00566
00567 // start catch_interfaces_testcase.h
00568
00569 #include <vector>
00570
00571 namespace Catch {
00572
00573     class TestSpec;

```

```

00571
00572     struct ITestInvoker {
00573         virtual void invoke () const = 0;
00574         virtual ~ITestInvoker();
00575     };
00576
00577     class TestCase;
00578     struct IConfig;
00579
00580     struct ITestCaseRegistry {
00581         virtual ~ITestCaseRegistry();
00582         virtual std::vector<TestCase> const& getAllTests() const = 0;
00583         virtual std::vector<TestCase> const& getAllTestsSorted( IConfig const& config ) const = 0;
00584     };
00585
00586     bool isThrowSafe( TestCase const& testCase, IConfig const& config );
00587     bool matchTest( TestCase const& testCase, TestSpec const& testSpec, IConfig const& config );
00588     std::vector<TestCase> filterTests( std::vector<TestCase> const& testCases, TestSpec const&
testSpec, IConfig const& config );
00589     std::vector<TestCase> const& getAllTestCasesSorted( IConfig const& config );
00590
00591 }
00592
00593 // end catch_interfaces_testcase.h
00594 // start catch_stringref.h
00595
00596 #include <cstddef>
00597 #include <string>
00598 #include <iosfwd>
00599 #include <cassert>
00600
00601 namespace Catch {
00602
00603     class StringRef {
00604     public:
00605         using size_type = std::size_t;
00606         using const_iterator = const char*;
00607
00608     private:
00609         static constexpr char const* s_empty = "";
00610
00611         char const* m_start = s_empty;
00612         size_type m_size = 0;
00613
00614     public: // construction
00615         constexpr StringRef() noexcept = default;
00616
00617         StringRef( char const* rawChars ) noexcept;
00618
00619         constexpr StringRef( char const* rawChars, size_type size ) noexcept
00620         :   m_start( rawChars ),
00621             m_size( size )
00622         {}
00623
00624         StringRef( std::string const& stdString ) noexcept
00625         :   m_start( stdString.c_str() ),
00626             m_size( stdString.size() )
00627         {}
00628
00629         explicit operator std::string() const {
00630             return std::string(m_start, m_size);
00631         }
00632
00633     public: // operators
00634         auto operator == ( StringRef const& other ) const noexcept -> bool;
00635         auto operator != ( StringRef const& other ) const noexcept -> bool {
00636             return !(*this == other);
00637         }
00638
00639         auto operator[] ( size_type index ) const noexcept -> char {
00640             assert(index < m_size);
00641             return m_start[index];
00642         }
00643
00644     public: // named queries
00645         constexpr auto empty() const noexcept -> bool {
00646             return m_size == 0;
00647         }
00648
00649         constexpr auto size() const noexcept -> size_type {
00650             return m_size;
00651         }
00652
00653         // Returns the current start pointer. If the StringRef is not
00654         // null-terminated, throws std::domain_exception
00655         auto c_str() const -> char const*;
00656
00657     public: // substrings and searches

```

```

00660         // Returns a substring of [start, start + length).
00661         // If start + length > size(), then the substring is [start, size()).
00662         // If start > size(), then the substring is empty.
00663         auto substr( size_type start, size_type length ) const noexcept -> StringRef;
00664
00665         // Returns the current start pointer. May not be null-terminated.
00666         auto data() const noexcept -> char const*;
00667
00668         constexpr auto isNullTerminated() const noexcept -> bool {
00669             return m_start[m_size] == '\0';
00670         }
00671
00672     public: // iterators
00673         constexpr const_iterator begin() const { return m_start; }
00674         constexpr const_iterator end() const { return m_start + m_size; }
00675     };
00676
00677     auto operator += ( std::string& lhs, StringRef const& sr ) -> std::string&;
00678     auto operator << ( std::ostream& os, StringRef const& sr ) -> std::ostream&;
00679
00680     constexpr auto operator "" _sr( char const* rawChars, std::size_t size ) noexcept -> StringRef {
00681         return StringRef( rawChars, size );
00682     }
00683 } // namespace Catch
00684
00685 constexpr auto operator "" _catch_sr( char const* rawChars, std::size_t size ) noexcept ->
Catch::StringRef {
00686     return Catch::StringRef( rawChars, size );
00687 }
00688
00689 // end catch_stringref.h
00690 // start catch_preprocessor.hpp
00691
00692 #define CATCH_RECursion_LEVEL0(...) __VA_ARGS__
00693 #define CATCH_RECursion_LEVEL1(...)
00694     CATCH_RECursion_LEVEL0(CATCH_RECursion_LEVEL0(CATCH_RECursion_LEVEL0(__VA_ARGS__)))
00695 #define CATCH_RECursion_LEVEL2(...)
00696     CATCH_RECursion_LEVEL1(CATCH_RECursion_LEVEL1(CATCH_RECursion_LEVEL1(__VA_ARGS__)))
00697 #define CATCH_RECursion_LEVEL3(...)
00698     CATCH_RECursion_LEVEL2(CATCH_RECursion_LEVEL2(CATCH_RECursion_LEVEL2(__VA_ARGS__)))
00699 #define CATCH_RECursion_LEVEL4(...)
00700     CATCH_RECursion_LEVEL3(CATCH_RECursion_LEVEL3(CATCH_RECursion_LEVEL3(__VA_ARGS__)))
00701 #define CATCH_RECursion_LEVEL5(...)
00702     CATCH_RECursion_LEVEL4(CATCH_RECursion_LEVEL4(CATCH_RECursion_LEVEL4(__VA_ARGS__)))
00703
00704 #ifdef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
00705 #define INTERNAL_CATCH_EXPAND_VARGS(...) __VA_ARGS__
00706 // MSVC needs more evaluations
00707 #define CATCH_RECursion_LEVEL6(...)
00708     CATCH_RECursion_LEVEL5(CATCH_RECursion_LEVEL5(CATCH_RECursion_LEVEL5(__VA_ARGS__)))
00709 #define CATCH_RECUSE(...) CATCH_RECursion_LEVEL6(CATCH_RECursion_LEVEL6(__VA_ARGS__))
00710 #else
00711 #define CATCH_RECUSE(...) CATCH_RECursion_LEVEL5(__VA_ARGS__)
00712 #endif
00713
00714 #define CATCH_REC_END(...)
00715 #define CATCH_REC_OUT
00716
00717 #define CATCH_EMPTY()
00718 #define CATCH_DEFER(id) id CATCH_EMPTY()
00719
00720 #define CATCH_REC_GET_END2() 0, CATCH_REC_END
00721 #define CATCH_REC_GET_END1(...) CATCH_REC_GET_END2
00722 #define CATCH_REC_GET_END(...) CATCH_REC_GET_END1
00723 #define CATCH_REC_NEXT0(test, next, ...) next CATCH_REC_OUT
00724 #define CATCH_REC_NEXT1(test, next) CATCH_DEFER ( CATCH_REC_NEXT0 ) ( test, next, 0)
00725 #define CATCH_REC_NEXT(test, next) CATCH_REC_NEXT1(CATCH_REC_GET_END test, next)
00726
00727 #define CATCH_REC_LIST0(f, x, peek, ...) , f(x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST1) )
00728 ( f, peek, __VA_ARGS__ )
00729 #define CATCH_REC_LIST1(f, x, peek, ...) , f(x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST0) )
00730 ( f, peek, __VA_ARGS__ )
00731 #define CATCH_REC_LIST2(f, x, peek, ...) f(x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST1) )
00732 ( f, peek, __VA_ARGS__ )
00733
00734 #define CATCH_REC_LIST0_UD(f, userdata, x, peek, ...) , f(userdata, x) CATCH_DEFER (
00735 CATCH_REC_NEXT(peek, CATCH_REC_LIST1_UD) ) ( f, userdata, peek, __VA_ARGS__ )
00736 #define CATCH_REC_LIST1_UD(f, userdata, x, peek, ...) , f(userdata, x) CATCH_DEFER (
00737 CATCH_REC_NEXT(peek, CATCH_REC_LIST0_UD) ) ( f, userdata, peek, __VA_ARGS__ )
00738 #define CATCH_REC_LIST2_UD(f, userdata, x, peek, ...) f(userdata, x) CATCH_DEFER (
00739 CATCH_REC_NEXT(peek, CATCH_REC_LIST1_UD) ) ( f, userdata, peek, __VA_ARGS__ )
00740
00741 // Applies the function macro `f` to each of the remaining parameters, inserts commas between the
00742 results,
00743 // and passes userdata as the first parameter to each invocation,
00744 // e.g. CATCH_REC_LIST_UD(f, x, a, b, c) evaluates to f(x, a), f(x, b), f(x, c)

```



```

00733 #define CATCH_REC_LIST_UD(f, userdata, ...) CATCH_RECURSE(CATCH_REC_LIST2_UD(f, userdata, __VA_ARGS__,
00734 () () (), () () (), () () (), 0))
00735 #define CATCH_REC_LIST(f, ...) CATCH_RECURSE(CATCH_REC_LIST2(f, __VA_ARGS__, () () (), () () (), () () (),
00736 0))
00737 #define INTERNAL_CATCH_EXPAND1(param) INTERNAL_CATCH_EXPAND2(param)
00738 #define INTERNAL_CATCH_EXPAND2(...) INTERNAL_CATCH_NO## __VA_ARGS__
00739 #define INTERNAL_CATCH_DEF(...) INTERNAL_CATCH_DEF __VA_ARGS__
00740 #define INTERNAL_CATCH_NOINTERNAL_CATCH_DEF
00741 #define INTERNAL_CATCH_STRINGIZE(...) INTERNAL_CATCH_STRINGIZE2(__VA_ARGS__)
00742 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
00743 #define INTERNAL_CATCH_STRINGIZE2(...) #__VA_ARGS__
00744 #define INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS(param)
INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_REMOVE_PARENS(param))
00745 #else
00746 // MSVC is adding extra space and needs another indirection to expand
INTERNAL_CATCH_DEF
00747 #define INTERNAL_CATCH_STRINGIZE2(...) INTERNAL_CATCH_STRINGIZE3(__VA_ARGS__)
00748 #define INTERNAL_CATCH_STRINGIZE3(...) #__VA_ARGS__
00749 #define INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS(param)
INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_REMOVE_PARENS(param)) + 1)
00750 #endif
00751
00752 #define INTERNAL_CATCH_MAKE_NAMESPACE2(...) ns_##__VA_ARGS__
00753 #define INTERNAL_CATCH_MAKE_NAMESPACE(name) INTERNAL_CATCH_MAKE_NAMESPACE2(name)
00754
00755 #define INTERNAL_CATCH_REMOVE_PARENS(...) INTERNAL_CATCH_EXPAND1(INTERNAL_CATCH_DEF __VA_ARGS__)
00756
00757 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
00758 #define INTERNAL_CATCH_MAKE_TYPE_LIST2(...)
decltype(get_wrapper<INTERNAL_CATCH_REMOVE_PARENS_GEN(__VA_ARGS__)>())
00759 #define INTERNAL_CATCH_MAKE_TYPE_LIST(...)
INTERNAL_CATCH_MAKE_TYPE_LIST2(INTERNAL_CATCH_REMOVE_PARENS(__VA_ARGS__))
00760 #else
00761 #define INTERNAL_CATCH_MAKE_TYPE_LIST2(...)
INTERNAL_CATCH_EXPAND_VARGS(decltype(get_wrapper<INTERNAL_CATCH_REMOVE_PARENS_GEN(__VA_ARGS__)>()))
00762 #define INTERNAL_CATCH_MAKE_TYPE_LIST(...)
INTERNAL_CATCH_EXPAND_VARGS(INTERNAL_CATCH_MAKE_TYPE_LIST2(INTERNAL_CATCH_REMOVE_PARENS(__VA_ARGS__)))
00763 #endif
00764
00765 #define INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(...) \
00766     CATCH_REC_LIST(INTERNAL_CATCH_MAKE_TYPE_LIST, __VA_ARGS__)
00767
00768 #define INTERNAL_CATCH_REMOVE_PARENS_1_ARG(_0) INTERNAL_CATCH_REMOVE_PARENS(_0)
00769 #define INTERNAL_CATCH_REMOVE_PARENS_2_ARG(_0, _1) INTERNAL_CATCH_REMOVE_PARENS(_0),
INTERNAL_CATCH_REMOVE_PARENS_1_ARG(_1)
00770 #define INTERNAL_CATCH_REMOVE_PARENS_3_ARG(_0, _1, _2) INTERNAL_CATCH_REMOVE_PARENS(_0),
INTERNAL_CATCH_REMOVE_PARENS_2_ARG(_1, _2)
00771 #define INTERNAL_CATCH_REMOVE_PARENS_4_ARG(_0, _1, _2, _3) INTERNAL_CATCH_REMOVE_PARENS(_0),
INTERNAL_CATCH_REMOVE_PARENS_3_ARG(_1, _2, _3)
00772 #define INTERNAL_CATCH_REMOVE_PARENS_5_ARG(_0, _1, _2, _3, _4) INTERNAL_CATCH_REMOVE_PARENS(_0),
INTERNAL_CATCH_REMOVE_PARENS_4_ARG(_1, _2, _3, _4)
00773 #define INTERNAL_CATCH_REMOVE_PARENS_6_ARG(_0, _1, _2, _3, _4, _5) INTERNAL_CATCH_REMOVE_PARENS(_0),
INTERNAL_CATCH_REMOVE_PARENS_5_ARG(_1, _2, _3, _4, _5)
00774 #define INTERNAL_CATCH_REMOVE_PARENS_7_ARG(_0, _1, _2, _3, _4, _5, _6)
INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_6_ARG(_1, _2, _3, _4, _5, _6)
00775 #define INTERNAL_CATCH_REMOVE_PARENS_8_ARG(_0, _1, _2, _3, _4, _5, _6, _7)
INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_7_ARG(_1, _2, _3, _4, _5, _6, _7)
00776 #define INTERNAL_CATCH_REMOVE_PARENS_9_ARG(_0, _1, _2, _3, _4, _5, _6, _7, _8)
INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_8_ARG(_1, _2, _3, _4, _5, _6, _7, _8)
00777 #define INTERNAL_CATCH_REMOVE_PARENS_10_ARG(_0, _1, _2, _3, _4, _5, _6, _7, _8, _9)
INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_9_ARG(_1, _2, _3, _4, _5, _6, _7, _8,
_9)
00778 #define INTERNAL_CATCH_REMOVE_PARENS_11_ARG(_0, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10)
INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_10_ARG(_1, _2, _3, _4, _5, _6, _7, _8,
_9, _10)
00779
00780 #define INTERNAL_CATCH_VA_NARGS_IMPL(_0, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, N, ...) N
00781
00782 #define INTERNAL_CATCH_TYPE_GEN \
00783     template<typename...> struct TypeList {};\
00784     template<typename...Ts>\
00785     constexpr auto get_wrapper() noexcept -> TypeList<Ts...> { return {}; }\
00786     template<template<typename...> class...> struct TemplateTypeList{};\
00787     template<template<typename...> class...Cs>\
00788     constexpr auto get_wrapper() noexcept -> TemplateTypeList<Cs...> { return {}; }\
00789     template<typename...>\
00790     struct append;\
00791     template<typename...>\
00792     struct rewrap;\
00793     template<template<typename...> class, typename...>\
00794     struct create;\
00795     template<template<typename...> class, typename>\
00796     struct convert;\
00797     \
00798     template<typename T> \

```



```

00799     struct append<T> { using type = T; };
00800     template< template<typename...> class L1, typename...E1, template<typename...> class L2,
typename...E2, typename...Rest>\
00801     struct append<L1<E1...>, L2<E2...>, Rest...> { using type = typename append<L1<E1...,E2...>,
Rest...>::type; };
00802     template< template<typename...> class L1, typename...E1, typename...Rest>\
00803     struct append<L1<E1...>, TypeList<mpl_::na>, Rest...> { using type = L1<E1...>; };
00804     \
00805     template< template<typename...> class Container, template<typename...> class List,
typename...elems>\
00806     struct rewrap<TemplateTypeList<Container>, List<elems...> > { using type =
TypeList<Container<elems...>; };
00807     template< template<typename...> class Container, template<typename...> class List, class...Elements>\
typename...Elements>\
00808     struct rewrap<TemplateTypeList<Container>, List<Elements...>, Elements...> { using type = typename
append<TypeList<Container<Elements...>, typename rewrap<TemplateTypeList<Container>,
Elements...>::type>::type; };
00809     \
00810     template<template <typename...> class Final, template< typename...> class...Containers,
typename...Types>\
00811     struct create<Final, TemplateTypeList<Containers...>, TypeList<Types...> > { using type = typename
append<Final<>, typename rewrap<TemplateTypeList<Containers>, Types...>::type...>::type; };
00812     template<template <typename...> class Final, template <typename...> class List, typename...Ts>\
00813     struct convert<Final, List<Ts...> > { using type = typename append<Final<>,TypeList<Ts>...>::type;
};
00814
00815 #define INTERNAL_CATCH_NTTP_1(signature, ...)\
00816     template<INTERNAL_CATCH_REMOVE_PARENS(signature)> struct Nttp{};\
00817     template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
00818     constexpr auto get_wrapper() noexcept -> Nttp<__VA_ARGS__> { return {}; } \
00819     template<template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class...> struct
NttpTemplateTypeList{};\
00820     template<template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class...Cs>\
00821     constexpr auto get_wrapper() noexcept -> NttpTemplateTypeList<Cs...> { return {}; } \
00822     \
00823     template< template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class Container,
template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class List,
INTERNAL_CATCH_REMOVE_PARENS(signature)>\
00824     struct rewrap<NttpTemplateTypeList<Container>, List<__VA_ARGS__> > { using type =
TypeList<Container<__VA_ARGS__>; };
00825     template< template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class Container,
template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class List, INTERNAL_CATCH_REMOVE_PARENS(signature),
typename...Elements>\
00826     struct rewrap<NttpTemplateTypeList<Container>, List<__VA_ARGS__>, Elements...> { using type =
typename append<TypeList<Container<__VA_ARGS__>, typename rewrap<NttpTemplateTypeList<Container>,
Elements...>::type>::type; };
00827     template<template <typename...> class Final, template<INTERNAL_CATCH_REMOVE_PARENS(signature)>
class...Containers, typename...Types>\
00828     struct create<Final, NttpTemplateTypeList<Containers...>, TypeList<Types...> > { using type =
typename append<Final<>, typename rewrap<NttpTemplateTypeList<Containers>, Types...>::type...>::type;
};
00829
00830 #define INTERNAL_CATCH_DECLARE_SIG_TEST0(TestName)
00831 #define INTERNAL_CATCH_DECLARE_SIG_TEST1(TestName, signature)\
00832     template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
00833     static void TestName()
00834 #define INTERNAL_CATCH_DECLARE_SIG_TEST_X(TestName, signature, ...)\
00835     template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
00836     static void TestName()
00837
00838 #define INTERNAL_CATCH_DEFINE_SIG_TEST0(TestName)
00839 #define INTERNAL_CATCH_DEFINE_SIG_TEST1(TestName, signature)\
00840     template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
00841     static void TestName()
00842 #define INTERNAL_CATCH_DEFINE_SIG_TEST_X(TestName, signature,...)\
00843     template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
00844     static void TestName()
00845
00846 #define INTERNAL_CATCH_NTTP_REGISTER0(TestFunc, signature)\
00847     template<typename Type>\
00848     void reg_test(TypeList<Type>, Catch::NameAndTags nameAndTags)\
00849     {\
00850         Catch::AutoReg( Catch::makeTestInvoker(&TestFunc<Type>), CATCH_INTERNAL_LINEINFO,
Catch::StringRef(), nameAndTags);\
00851     }
00852
00853 #define INTERNAL_CATCH_NTTP_REGISTER(TestFunc, signature, ...)\
00854     template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
00855     void reg_test(Nttp<__VA_ARGS__>, Catch::NameAndTags nameAndTags)\
00856     {\
00857         Catch::AutoReg( Catch::makeTestInvoker(&TestFunc<__VA_ARGS__>), CATCH_INTERNAL_LINEINFO,
Catch::StringRef(), nameAndTags);\
00858     }
00859
00860 #define INTERNAL_CATCH_NTTP_REGISTER_METHOD0(TestName, signature, ...)\
00861     template<typename Type>\
00862     void reg_test(TypeList<Type>, Catch::StringRef className, Catch::NameAndTags nameAndTags)\

```



```

INTERNAL_CATCH_NTTP_1, INTERNAL_CATCH_NTTP_1, INTERNAL_CATCH_NTTP_1,
INTERNAL_CATCH_NTTP_1, INTERNAL_CATCH_NTTP_1, INTERNAL_CATCH_NTTP_0) ( __VA_ARGS__ )
00908 #define INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD(TestName, ...)
INTERNAL_CATCH_EXPAND_VARGS(INTERNAL_CATCH_VA_NARGS_IMPL( "dummy", __VA_ARGS__,
INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD1, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD0)(TestName,
__VA_ARGS__))
00909 #define INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD(TestName, ClassName, ...)
INTERNAL_CATCH_EXPAND_VARGS(INTERNAL_CATCH_VA_NARGS_IMPL( "dummy", __VA_ARGS__,
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD1, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD0)(TestName, ClassName,
__VA_ARGS__))
00910 #define INTERNAL_CATCH_NTTP_REG_METHOD_GEN(TestName, ...)
INTERNAL_CATCH_EXPAND_VARGS(INTERNAL_CATCH_VA_NARGS_IMPL( "dummy", __VA_ARGS__,
INTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD,
INTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD,
INTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD,
INTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD0,
INTERNAL_CATCH_NTTP_REGISTER_METHOD0)(TestName, __VA_ARGS__))
00911 #define INTERNAL_CATCH_NTTP_REG_GEN(TestFunc, ...)
INTERNAL_CATCH_EXPAND_VARGS(INTERNAL_CATCH_VA_NARGS_IMPL( "dummy", __VA_ARGS__,
INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER,
INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER,
INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER,
INTERNAL_CATCH_NTTP_REGISTER0, INTERNAL_CATCH_NTTP_REGISTER0)(TestFunc, __VA_ARGS__))
00912 #define INTERNAL_CATCH_DEFINE_SIG_TEST(TestName, ...)
INTERNAL_CATCH_EXPAND_VARGS(INTERNAL_CATCH_VA_NARGS_IMPL( "dummy", __VA_ARGS__,
INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST1,
INTERNAL_CATCH_DEFINE_SIG_TEST0)(TestName, __VA_ARGS__))
00913 #define INTERNAL_CATCH_DECLARE_SIG_TEST(TestName, ...)
INTERNAL_CATCH_EXPAND_VARGS(INTERNAL_CATCH_VA_NARGS_IMPL( "dummy", __VA_ARGS__,
INTERNAL_CATCH_DECLARE_SIG_TEST_X, INTERNAL_CATCH_DECLARE_SIG_TEST_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_X, INTERNAL_CATCH_DECLARE_SIG_TEST_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_X, INTERNAL_CATCH_DECLARE_SIG_TEST_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_X, INTERNAL_CATCH_DECLARE_SIG_TEST_X,
INTERNAL_CATCH_DECLARE_SIG_TEST1, INTERNAL_CATCH_DECLARE_SIG_TEST0)(TestName, __VA_ARGS__))
00914 #define INTERNAL_CATCH_REMOVE_PARENS_GEN(...)
INTERNAL_CATCH_EXPAND_VARGS(INTERNAL_CATCH_VA_NARGS_IMPL(__VA_ARGS__,
INTERNAL_CATCH_REMOVE_PARENS_11_ARG, INTERNAL_CATCH_REMOVE_PARENS_10_ARG, INTERNAL_CATCH_REMOVE_PARENS_9_ARG, INTERNAL_CATCH_REMOVE_PARENS_8_ARG,
INTERNAL_CATCH_REMOVE_PARENS_7_ARG, INTERNAL_CATCH_REMOVE_PARENS_6_ARG, INTERNAL_CATCH_REMOVE_PARENS_5_ARG, INTERNAL_CATCH_REMOVE_PARENS_4_ARG,
INTERNAL_CATCH_REMOVE_PARENS_3_ARG, INTERNAL_CATCH_REMOVE_PARENS_2_ARG, INTERNAL_CATCH_REMOVE_PARENS_1_ARG, INTERNAL_CATCH_REMOVE_PARENS_0_ARG)...)
00915 #endif
00916
00917 // end catch_preprocessor.hpp
00918 // start catch_meta.hpp
00919
00920
00921 #include <type_traits>
00922
00923 namespace Catch {
00924     template<typename T>
00925     struct always_false : std::false_type {};
00926
00927     template <typename> struct true_given : std::true_type {};
00928     struct is_callable_tester {
00929         template <typename Fun, typename... Args>
00930         true_given<decltype(std::declval<Fun>() (std::declval<Args>()...))> static test(int);
00931         template <typename...>
00932         std::false_type static test(...);
00933     };
00934
00935     template <typename T>
00936     struct is_callable;
00937
00938     template <typename Fun, typename... Args>
00939     struct is_callable<Fun(Args...)> : decltype(is_callable_tester::test<Fun, Args...>(0)) {};
00940
00941 #if defined(__cpp_lib_is_invocable) && __cpp_lib_is_invocable >= 201703
00942     // std::result_of is deprecated in C++17 and removed in C++20. Hence, it is
00943     // replaced with std::invoke_result here.
00944     template <typename Func, typename... U>
00945     using FunctionReturnType = std::remove_reference_t<std::remove_cv_t<std::invoke_result_t<Func,
U...>>>;
00946 #else
00947     // Keep ::type here because we still support C++11
00948     template <typename Func, typename... U>
00949     using FunctionReturnType = typename std::remove_reference<typename std::remove_cv<typename
std::result_of<Func(U...)>::type>::type>::type;
00950 #endif

```



```

INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, Signature, __VA_ARGS__ ) )
01027     #endif
01028
01029     #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01030         #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION( ClassName, Name, Tags,... )
\
01031             INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION_2( INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_C_L_A_S_S_ ), INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), ClassName, Name, Tags, typename T, __VA_ARGS__ )
01032     #else
01033         #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION( ClassName, Name, Tags,... )
\
01034             INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION_2(
INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_C_L_A_S_S_ ), INTERNAL_CATCH_UNIQUE_NAME(
INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), ClassName, Name, Tags, typename T,
__VA_ARGS__ ) ) )
01035     #endif
01036
01037     #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01038         #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION( ClassName, Name, Tags,
Signature, ... ) \
01039             INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION_2( INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_C_L_A_S_S_ ), INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), ClassName, Name, Tags, Signature, __VA_ARGS__ )
01040     #else
01041         #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION( ClassName, Name, Tags,
Signature, ... ) \
01042             INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION_2(
INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_C_L_A_S_S_ ), INTERNAL_CATCH_UNIQUE_NAME(
INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), ClassName, Name, Tags, Signature,
__VA_ARGS__ ) ) )
01043     #endif
01044 #endif
01045
01046     #define INTERNAL_CATCH_TESTCASE2( TestName, ... ) \
01047         static void TestName(); \
01048         CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01049         CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01050         namespace{ Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar )( Catch::makeTestInvoker(
&TestName ), CATCH_INTERNAL_LINEINFO, Catch::StringRef(), Catch::NameAndTags{ __VA_ARGS__ } ); } /*
NOLINT */ \
01051         CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
01052         static void TestName()
01053     #define INTERNAL_CATCH_TESTCASE( ... ) \
01054         INTERNAL_CATCH_TESTCASE2( INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ), __VA_ARGS__ )
01055
01056     #define INTERNAL_CATCH_METHOD_AS_TEST_CASE( QualifiedMethod, ... ) \
01057         CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01058         CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01059         namespace{ Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar )( Catch::makeTestInvoker(
&QualifiedMethod ), CATCH_INTERNAL_LINEINFO, "&" #QualifiedMethod, Catch::NameAndTags{ __VA_ARGS__ }
); } /* NOLINT */ \
01060         CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
01061
01062     #define INTERNAL_CATCH_TEST_CASE_METHOD2( TestName, ClassName, ... )\
01063         CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01064         CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01065         namespace{ \
01066             struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName) { \
01067                 void test(); \
01068             }; \
01069             Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar ) ( Catch::makeTestInvoker(
&TestName::test ), CATCH_INTERNAL_LINEINFO, #ClassName, Catch::NameAndTags{ __VA_ARGS__ } ); /* NOLINT
*/ \
01070         } \
01071         CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
01072         void TestName::test()
01073     #define INTERNAL_CATCH_TEST_CASE_METHOD( ClassName, ... ) \
01074         INTERNAL_CATCH_TEST_CASE_METHOD2( INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ), ClassName,
__VA_ARGS__ )
01075
01076     #define INTERNAL_CATCH_REGISTER_TESTCASE( Function, ... ) \
01077         CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01078         CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01079         Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar )( Catch::makeTestInvoker( Function
), CATCH_INTERNAL_LINEINFO, Catch::StringRef(), Catch::NameAndTags{ __VA_ARGS__ } ); /* NOLINT */ \
01080         CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
01081
01082     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_2( TestName, TestFunc, Name, Tags, Signature, ... )\
01083         CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01084         CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01085         CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
01086         CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
01087         INTERNAL_CATCH_DECLARE_SIG_TEST( TestFunc, INTERNAL_CATCH_REMOVE_PARENS( Signature ) ); \
01088         namespace { \
01089             namespace INTERNAL_CATCH_MAKE_NAMESPACE( TestName ) {

```

```

01095         INTERNAL_CATCH_TYPE_GEN\
01096         INTERNAL_CATCH_NTTP_GEN(INTERNAL_CATCH_REMOVE_PARENS(Signature))\
01097         INTERNAL_CATCH_NTTP_REG_GEN(TestFunc,INTERNAL_CATCH_REMOVE_PARENS(Signature))\
01098         template<typename...Types> \
01099         struct TestName{\
01100             TestName(){\
01101                 int index = 0;\
01102                 constexpr char const* tmp_types[] = \
01103 {CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, __VA_ARGS__)};\
01104                 using expander = int[];\
01105                 (void)expander{(reg_test(Types{}, Catch::NameAndTags{ Name " - " +
std::string(tmp_types[index]), Tags } ), index++)... };/* NOLINT */ \
01106             }\
01107             static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){\
01108                 TestName<INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(__VA_ARGS__)>();\
01109                 return 0;\
01110             }();\
01111         }\
01112     }\
01113     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
01114     INTERNAL_CATCH_DEFINE_SIG_TEST(TestFunc,INTERNAL_CATCH_REMOVE_PARENS(Signature))
01115
01116 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01117     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE(Name, Tags, ...) \
01118         INTERNAL_CATCH_TEMPLATE_TEST_CASE_2( INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, typename TestType, __VA_ARGS__ )
01119 #else
01120     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE(Name, Tags, ...) \
01121         INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_2( INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, typename TestType, __VA_ARGS__ ) )
01122 #endif
01123
01124 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01125     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG(Name, Tags, Signature, ...) \
01126         INTERNAL_CATCH_TEMPLATE_TEST_CASE_2( INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, Signature, __VA_ARGS__ )
01127 #else
01128     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG(Name, Tags, Signature, ...) \
01129         INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_2( INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, Signature, __VA_ARGS__ ) )
01130 #endif
01131
01132     #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(TestName, TestFuncName, Name, Tags, Signature,
TmplTypes, TypesList) \
01133         CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01134         CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01135         CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
01136         CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
01137         template<typename TestType> static void TestFuncName(); \
01138         namespace {\
01139             namespace INTERNAL_CATCH_MAKE_NAMESPACE(TestName) { \
01140                 INTERNAL_CATCH_TYPE_GEN \
01141                 INTERNAL_CATCH_NTTP_GEN(INTERNAL_CATCH_REMOVE_PARENS(Signature)) \
01142                 template<typename... Types> \
01143                 struct TestName { \
01144                     void reg_tests() { \
01145                         int index = 0;\
01146                         using expander = int[];\
01147                         constexpr char const* tmp_types[] = \
01148 {CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, INTERNAL_CATCH_REMOVE_PARENS(TmplTypes))};\
01149                         constexpr char const* types_list[] = \
01150 {CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, INTERNAL_CATCH_REMOVE_PARENS(TypesList))};\
01151                         constexpr auto num_types = sizeof(types_list) / sizeof(types_list[0]);\
01152                         (void)expander{(Catch::AutoReg( Catch::makeTestInvoker( &TestFuncName<Types> ),
CATCH_INTERNAL_LINEINFO, Catch::StringRef(), Catch::NameAndTags{ Name " - " +
std::string(tmp_types[index / num_types]) + "<" + std::string(types_list[index % num_types]) + ">",
Tags } ), index++)... };/* NOLINT */\
01153                     }\
01154                     static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){ \
01155                         using TestInit = typename create<TestName,\
decltype(get_wrapper<INTERNAL_CATCH_REMOVE_PARENS(TmplTypes)>())\
TypeList<INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(INTERNAL_CATCH_REMOVE_PARENS(TypesList))>::type; \
01156                         TestInit t;\
01157                         t.reg_tests();\
01158                         return 0;\
01159                     }(); \
01160                 }\
01161                 CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
01162                 template<typename TestType> \
01163                 static void TestFuncName()

```



```

01164
01165 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01166     #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE(Name, Tags, ...) \
01167         INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(INTERNAL_CATCH_UNIQUE_NAME(
01168             C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
01169             C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, typename T, __VA_ARGS__)
01170 #else
01171     #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE(Name, Tags, ...) \
01172         INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(
01173             INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
01174             C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, typename T, __VA_ARGS__ ) ) )
01175 #endif
01176
01177 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01178     #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG(Name, Tags, Signature, ...) \
01179         INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(INTERNAL_CATCH_UNIQUE_NAME(
01180             C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
01181             C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, Signature, __VA_ARGS__)
01182 #else
01183     #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG(Name, Tags, Signature, ...) \
01184         INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(
01185             INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
01186             C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, Signature, __VA_ARGS__ ) ) )
01187 #endif
01188
01189 #define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_2( TestName, TestFunc, Name, Tags, TmplList ) \
01190     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01191     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01192     CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
01193     template<typename TestType> static void TestFunc(); \
01194     namespace { \
01195         namespace INTERNAL_CATCH_MAKE_NAMESPACE( TestName ) { \
01196             INTERNAL_CATCH_TYPE_GEN \
01197             template<typename... Types> \
01198             struct TestName { \
01199                 void reg_tests() { \
01200                     int index = 0; \
01201                     using expander = int[]; \
01202                     (void)expander{(Catch::AutoReg( Catch::makeTestInvoker( &TestFunc<Types> ), \
01203                         CATCH_INTERNAL_LINEINFO, Catch::StringRef(), Catch::NameAndTags{ Name " - " + \
01204                             std::string(INTERNAL_CATCH_STRINGIZE(TmplList)) + " - " + std::to_string(index), Tags } ), index++)... \
01205                     };/* NOLINT */ \
01206                 } \
01207             }; \
01208             static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = []() { \
01209                 using TestInit = typename convert<TestName, TmplList>::type; \
01210                 TestInit t; \
01211                 t.reg_tests(); \
01212                 return 0; \
01213             }(); \
01214         } \
01215     } \
01216     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
01217     template<typename TestType> \
01218     static void TestFunc()
01219
01220 #define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE(Name, Tags, TmplList) \
01221     INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_2( INTERNAL_CATCH_UNIQUE_NAME( \
01222         C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME( \
01223         C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, TmplList )
01224
01225 #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_2( TestNameClass, TestName, ClassName, Name, \
01226     Tags, Signature, ... ) \
01227     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01228     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01229     CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
01230     CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
01231     namespace { \
01232         namespace INTERNAL_CATCH_MAKE_NAMESPACE( TestName ) { \
01233             INTERNAL_CATCH_TYPE_GEN \
01234             INTERNAL_CATCH_NTTP_GEN(INTERNAL_CATCH_REMOVE_PARENS( Signature )) \
01235             INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD( TestName, ClassName, \
01236                 INTERNAL_CATCH_REMOVE_PARENS( Signature )) \
01237             INTERNAL_CATCH_NTTP_REG_METHOD_GEN( TestName, INTERNAL_CATCH_REMOVE_PARENS( Signature )) \
01238             template<typename... Types> \
01239             struct TestNameClass { \
01240                 TestNameClass() { \
01241                     int index = 0; \
01242                     constexpr char const* tmpl_types[] = \
01243                         {CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, __VA_ARGS__)}; \
01244                     using expander = int[]; \
01245                     (void)expander{(reg_test( Types{}, #ClassName, Catch::NameAndTags{ Name " - " + \
01246                         std::string( tmpl_types[index] ), Tags } ), index++)... };/* NOLINT */ \
01247                 } \
01248             }; \
01249             static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = []() { \
01250                 TestNameClass<INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(__VA_ARGS__)>(); \
01251                 return 0; \
01252             }(); \
01253         } \
01254     }

```

```

01234         }();\
01235     }\
01236 }\
01237     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
01238     INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD(TestName, INTERNAL_CATCH_REMOVE_PARENS(Signature))
01239
01240 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01241     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD( ClassName, Name, Tags,... ) \
01242     INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_2( INTERNAL_CATCH_UNIQUE_NAME(
01243         C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_C_L_A_S_S_ ), INTERNAL_CATCH_UNIQUE_NAME(
01244         C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), ClassName, Name, Tags, typename T, __VA_ARGS__ )
01245 #else
01246     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD( ClassName, Name, Tags,... ) \
01247     INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_2(
01248     INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_C_L_A_S_S_ ),
01249     INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), ClassName, Name, Tags, typename T,
01250     __VA_ARGS__ ) )
01251 #endif
01252
01253 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01254     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( ClassName, Name, Tags, Signature, ... ) \
01255     INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_2( INTERNAL_CATCH_UNIQUE_NAME(
01256     C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_C_L_A_S_S_ ), INTERNAL_CATCH_UNIQUE_NAME(
01257     C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), ClassName, Name, Tags, Signature, __VA_ARGS__ )
01258 #else
01259     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( ClassName, Name, Tags, Signature, ... ) \
01260     INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_2(
01261     INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_C_L_A_S_S_ ),
01262     INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), ClassName, Name, Tags, Signature,
01263     __VA_ARGS__ ) )
01264 #endif
01265
01266 #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_2( TestNameClass, TestName, ClassName,
01267     Name, Tags, Signature, TmplTypes, TypesList)\
01268     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01269     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01270     CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
01271     CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
01272     template<typename TestType> \
01273     struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName <TestType>) { \
01274         void test(); \
01275     }; \
01276     namespace {\
01277     namespace INTERNAL_CATCH_MAKE_NAMESPACED(TestNameClass) {\
01278         INTERNAL_CATCH_TYPE_GEN \
01279         INTERNAL_CATCH_NTTP_GEN(INTERNAL_CATCH_REMOVE_PARENS(Signature))\
01280         template<typename...Types>\
01281         struct TestNameClass{\
01282             void reg_tests(){\
01283                 int index = 0;\
01284                 using expander = int[];\
01285                 constexpr char const* tmpl_types[] =
01286 {CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, INTERNAL_CATCH_REMOVE_PARENS(TmplTypes))};\
01287                 constexpr char const* types_list[] =
01288 {CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, INTERNAL_CATCH_REMOVE_PARENS(TypesList))};\
01289                 constexpr auto num_types = sizeof(types_list) / sizeof(types_list[0]);\
01290                 (void)expander{(Catch::AutoReg( Catch::makeTestInvoker( &TestName<Types>::test ),
01291 CATCH_INTERNAL_LINEINFO, #ClassName, Catch::NameAndTags{ Name " - " + std::string(tmpl_types[index /
01292 num_types]) + "<" + std::string(types_list[index % num_types]) + ">", Tags } ), index++)... };\
01293                 NOLINT */ \
01294             }\
01295         }; \
01296         static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){\
01297             using TestInit = typename create<TestNameClass,
01298 decltype(get_wrapper<INTERNAL_CATCH_REMOVE_PARENS(TmplTypes)>()),
01299 TypeList<INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(INTERNAL_CATCH_REMOVE_PARENS(TypesList))>::type;\
01300             TestInit t;\
01301             t.reg_tests(); \
01302             return 0;\
01303         }(); \
01304     }\
01305     }\
01306     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
01307     template<typename TestType> \
01308     void TestName<TestType>::test()
01309
01310 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01311     #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( ClassName, Name, Tags, ... )\
01312     INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_2( INTERNAL_CATCH_UNIQUE_NAME(
01313     C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
01314     C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), ClassName, Name, Tags, typename T, __VA_ARGS__ )
01315 #else
01316     #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( ClassName, Name, Tags, ... )\
01317     INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_2(
01318     INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
01319     C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), ClassName, Name, Tags, typename T, __VA_ARGS__ ) )
01320 #endif

```



```

01299
01300 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01301     #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( ClassName, Name, Tags, Signature,
... )\
01302         INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_2( INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), ClassName, Name, Tags, Signature, __VA_ARGS__ )
01303 #else
01304     #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( ClassName, Name, Tags, Signature,
... )\
01305         INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_2(
INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), ClassName, Name, Tags, Signature, __VA_ARGS__ ) )
01306 #endif
01307
01308     #define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD_2( TestNameClass, TestName, ClassName, Name,
Tags, TmplList) \
01309         CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01310         CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01311         CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
01312         template<typename TestType> \
01313         struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName <TestType>) { \
01314             void test(); \
01315         }; \
01316         namespace { \
01317             namespace INTERNAL_CATCH_MAKE_NAMESPACE( TestName ) { \
01318                 INTERNAL_CATCH_TYPE_GEN \
01319                 template<typename... Types> \
01320                 struct TestNameClass { \
01321                     void reg_tests() { \
01322                         int index = 0; \
01323                         using expander = int[]; \
01324                         (void)expander{ (Catch::AutoReg( Catch::makeTestInvoker( &TestName<Types>::test ),
CATCH_INTERNAL_LINEINFO, #ClassName, Catch::NameAndTags{ Name " - " +
std::string(INTERNAL_CATCH_STRINGIZE(TmplList)) + " - " + std::to_string(index), Tags } ), index++)...
}; /* NOLINT */ \
01325                     } \
01326                 }; \
01327                 static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = []() { \
01328                     using TestInit = typename convert<TestNameClass, TmplList>::type; \
01329                     TestInit t; \
01330                     t.reg_tests(); \
01331                     return 0; \
01332                 }(); \
01333             } \
01334             CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
01335             template<typename TestType> \
01336             void TestName<TestType>::test()
01337
01338     #define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD(ClassName, Name, Tags, TmplList) \
01339         INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD_2( INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), ClassName, Name, Tags, TmplList )
01340
01341 // end catch_test_registry.h
01342 // start catch_capture.hpp
01343
01344 // start catch_assertionhandler.h
01345
01346 // start catch_assertioninfo.h
01347
01348 // start catch_result_type.h
01349
01350 namespace Catch {
01351
01352     // ResultWas::OfType enum
01353     struct ResultWas { enum OfType {
01354         Unknown = -1,
01355         Ok = 0,
01356         Info = 1,
01357         Warning = 2,
01358
01359         FailureBit = 0x10,
01360
01361         ExpressionFailed = FailureBit | 1,
01362         ExplicitFailure = FailureBit | 2,
01363
01364         Exception = 0x100 | FailureBit,
01365
01366         ThrewException = Exception | 1,
01367         DidntThrowException = Exception | 2,
01368
01369         FatalErrorCondition = 0x200 | FailureBit
01370     }; };
01371
01372     bool isOk( ResultWas::OfType resultType );

```

```

01374     bool isJustInfo( int flags );
01375
01376     // ResultDisposition::Flags enum
01377     struct ResultDisposition { enum Flags {
01378         Normal = 0x01,
01379
01380         ContinueOnFailure = 0x02,    // Failures fail test, but execution continues
01381         FalseTest = 0x04,            // Prefix expression with !
01382         SuppressFail = 0x08          // Failures are reported but do not fail the test
01383     }; };
01384
01385     ResultDisposition::Flags operator | ( ResultDisposition::Flags lhs, ResultDisposition::Flags rhs
);
01386
01387     bool shouldContinueOnFailure( int flags );
01388     inline bool isFalseTest( int flags ) { return ( flags & ResultDisposition::FalseTest ) != 0; }
01389     bool shouldSuppressFailure( int flags );
01390
01391 } // end namespace Catch
01392
01393 // end catch_result_type.h
01394 namespace Catch {
01395
01396     struct AssertionInfo
01397     {
01398        StringRef macroName;
01399        SourceLineInfo lineInfo;
01400        StringRef capturedExpression;
01401        ResultDisposition::Flags resultDisposition;
01402
01403         // We want to delete this constructor but a compiler bug in 4.8 means
01404         // the struct is then treated as non-aggregate
01405         //AssertionInfo() = delete;
01406     };
01407
01408 } // end namespace Catch
01409
01410 // end catch_assertioninfo.h
01411 // start catch_decomposer.h
01412
01413 // start catch_tostring.h
01414
01415 #include <vector>
01416 #include <cstdint>
01417 #include <type_traits>
01418 #include <string>
01419 // start catch_stream.h
01420
01421 #include <iosfwd>
01422 #include <cstdint>
01423 #include <ostream>
01424
01425 namespace Catch {
01426
01427     std::ostream& cout();
01428     std::ostream& cerr();
01429     std::ostream& clog();
01430
01431     class StringRef;
01432
01433     struct IStream {
01434         virtual ~IStream();
01435         virtual std::ostream& stream() const = 0;
01436     };
01437
01438     auto makeStream( StringRef const &filename ) -> IStream const*;
01439
01440     class ReusableStringStream : NonCopyable {
01441     public:
01442         std::size_t m_index;
01443         std::ostream* m_oss;
01444         ReusableStringStream();
01445         ~ReusableStringStream();
01446
01447         auto str() const -> std::string;
01448
01449         template<typename T>
01450         auto operator << ( T const& value ) -> ReusableStringStream& {
01451             *m_oss << value;
01452             return *this;
01453         }
01454         auto get() -> std::ostream& { return *m_oss; }
01455     };
01456 }
01457
01458 // end catch_stream.h
01459 // start catch_interfaces_enum_values_registry.h

```

```

01460
01461 #include <vector>
01462
01463 namespace Catch {
01464
01465     namespace Detail {
01466         struct EnumInfo {
01467            StringRef m_name;
01468             std::vector<std::pair<int, StringRef>> m_values;
01469
01470             ~EnumInfo();
01471
01472             StringRef lookup( int value ) const;
01473         };
01474     } // namespace Detail
01475
01476     struct IMutableEnumValuesRegistry {
01477         virtual ~IMutableEnumValuesRegistry();
01478
01479         virtual Detail::EnumInfo const& registerEnum( StringRef enumName, StringRef allEnums,
01480             std::vector<int> const& values ) = 0;
01481
01482         template<typename E>
01483         Detail::EnumInfo const& registerEnum( StringRef enumName, StringRef allEnums,
01484             std::initializer_list<E> values ) {
01485             static_assert(sizeof(int) >= sizeof(E), "Cannot serialize enum to int");
01486             std::vector<int> intValues;
01487             intValues.reserve( values.size() );
01488             for( auto enumValue : values )
01489                 intValues.push_back( static_cast<int>( enumValue ) );
01490             return registerEnum( enumName, allEnums, intValues );
01491         }
01492     };
01493
01494 } // Catch
01495
01496 // end catch_interfaces_enum_values_registry.h
01497
01498 #ifdef CATCH_CONFIG_CPP17_STRING_VIEW
01499 #include <string_view>
01500 #endif
01501
01502 #ifdef __OBJC__
01503 // start catch_objc_arc.hpp
01504
01505 #import <Foundation/Foundation.h>
01506
01507 #ifdef __has_feature
01508 #define CATCH_ARC_ENABLED __has_feature(objc_arc)
01509 #else
01510 #define CATCH_ARC_ENABLED 0
01511 #endif
01512
01513 void arcSafeRelease( NSObject* obj );
01514 id performOptionalSelector( id obj, SEL sel );
01515
01516 #if !CATCH_ARC_ENABLED
01517 inline void arcSafeRelease( NSObject* obj ) {
01518     [obj release];
01519 }
01520 inline id performOptionalSelector( id obj, SEL sel ) {
01521     if( [obj respondsToSelector: sel] )
01522         return [obj performSelector: sel];
01523     return nil;
01524 }
01525 #define CATCH_UNSAFE_UNRETAINED
01526 #define CATCH_ARC_STRONG
01527 #else
01528 inline void arcSafeRelease( NSObject* ){}
01529 inline id performOptionalSelector( id obj, SEL sel ) {
01530     #ifdef __clang__
01531     #pragma clang diagnostic push
01532     #pragma clang diagnostic ignored "-Warc-performSelector-leaks"
01533     #endif
01534     if( [obj respondsToSelector: sel] )
01535         return [obj performSelector: sel];
01536     #ifdef __clang__
01537     #pragma clang diagnostic pop
01538     #endif
01539     return nil;
01540 }
01541 #define CATCH_UNSAFE_UNRETAINED __unsafe_unretained
01542 #define CATCH_ARC_STRONG __strong
01543 #endif
01544 // end catch_objc_arc.hpp
01545 #endif

```

```

01545
01546 #ifndef _MSC_VER
01547 #pragma warning(push)
01548 #pragma warning(disable:4180) // We attempt to stream a function (address) by const&, which MSVC
                                complains about but is harmless
01549 #endif
01550
01551 namespace Catch {
01552     namespace Detail {
01553
01554         extern const std::string unprintableString;
01555
01556         std::string rawMemoryToString( const void *object, std::size_t size );
01557
01558         template<typename T>
01559         std::string rawMemoryToString( const T& object ) {
01560             return rawMemoryToString( &object, sizeof(object) );
01561         }
01562
01563         template<typename T>
01564         class IsStreamInsertable {
01565             template<typename Stream, typename U>
01566             static auto test(int)
01567                 -> decltype(std::declval<Stream&>() < std::declval<U>(), std::true_type());
01568
01569             template<typename, typename>
01570             static auto test(...) -> std::false_type;
01571
01572         public:
01573             static const bool value = decltype(test<std::ostream, const T&>(0))::value;
01574         };
01575
01576         template<typename E>
01577         std::string convertUnknownEnumToString( E e );
01578
01579         template<typename T>
01580         typename std::enable_if<
01581             !std::is_enum<T>::value && !std::is_base_of<std::exception, T>::value,
01582             std::string>::type convertUnstreamable( T const& ) {
01583             return Detail::unprintableString;
01584         }
01585         template<typename T>
01586         typename std::enable_if<
01587             !std::is_enum<T>::value && std::is_base_of<std::exception, T>::value,
01588             std::string>::type convertUnstreamable( T const& ex ) {
01589             return ex.what();
01590         }
01591
01592         template<typename T>
01593         typename std::enable_if<
01594             std::is_enum<T>::value
01595             , std::string>::type convertUnstreamable( T const& value ) {
01596             return convertUnknownEnumToString( value );
01597         }
01598
01599 #if defined(_MANAGED)
01600         template<typename T>
01601         std::string clrReferenceToString( T^ ref ) {
01602             if (ref == nullptr)
01603                 return std::string("null");
01604             auto bytes = System::Text::Encoding::UTF8->GetBytes(ref->ToString());
01605             cli::pin_ptr<System::Byte> p = &bytes[0];
01606             return std::string(reinterpret_cast<char const *>(p), bytes->Length);
01607         }
01608 #endif
01609     } // namespace Detail
01610 } // namespace Catch
01611
01612 // If we decide for C++14, change these to enable_if_ts
01613 template <typename T, typename = void>
01614 struct StringMaker {
01615     template <typename Fake = T>
01616     static
01617     typename std::enable_if<!: Catch::Detail::IsStreamInsertable<Fake>::value, std::string>::type
01618         convert(const Fake& value) {
01619         ReusableStringStream rss;
01620         // NB: call using the function-like syntax to avoid ambiguity with
01621         // user-defined templated operator< under clang.
01622         rss.operator<<(value);
01623         return rss.str();
01624     }
01625
01626     template <typename Fake = T>
01627     static
01628     typename std::enable_if<!: Catch::Detail::IsStreamInsertable<Fake>::value, std::string>::type
01629         convert( const Fake& value ) {
01630     }
01631 #if !defined(CATCH_CONFIG_FALLBACK_STRINGIFIER)

```

```

01632         return Detail::convertUnstreamable(value);
01633     #else
01634         return CATCH_CONFIG_FALLBACK_STRINGIFIER(value);
01635     #endif
01636     }
01637 };
01638
01639 namespace Detail {
01640
01641     // This function dispatches all stringification requests inside of Catch.
01642     // Should be preferably called fully qualified, like ::Catch::Detail::stringify
01643     template <typename T>
01644     std::string stringify(const T& e) {
01645         return ::Catch::StringMaker<typename std::remove_cv<typename
std::remove_reference<T>::type>::type>::convert(e);
01646     }
01647
01648     template<typename E>
01649     std::string convertUnknownEnumToString( E e ) {
01650         return ::Catch::Detail::stringify(static_cast<typename std::underlying_type<E>::type>(e));
01651     }
01652
01653     #if defined(_MANAGED)
01654     template <typename T>
01655     std::string stringify( T^ e ) {
01656         return ::Catch::StringMaker<T^>::convert(e);
01657     }
01658     #endif
01659 } // namespace Detail
01660
01661 // Some predefined specializations
01662
01663 template<>
01664 struct StringMaker<std::string> {
01665     static std::string convert(const std::string& str);
01666 };
01667
01668 #ifndef CATCH_CONFIG_CPP17_STRING_VIEW
01669 template<>
01670 struct StringMaker<std::string_view> {
01671     static std::string convert(std::string_view str);
01672 };
01673 #endif
01674
01675 template<>
01676 struct StringMaker<char const*> {
01677     static std::string convert(char const* str);
01678 };
01679
01680 template<>
01681 struct StringMaker<char*> {
01682     static std::string convert(char* str);
01683 };
01684
01685 #ifndef CATCH_CONFIG_WCHAR
01686 template<>
01687 struct StringMaker<std::wstring> {
01688     static std::string convert(const std::wstring& wstr);
01689 };
01690
01691 # ifdef CATCH_CONFIG_CPP17_STRING_VIEW
01692 template<>
01693 struct StringMaker<std::wstring_view> {
01694     static std::string convert(std::wstring_view str);
01695 };
01696 # endif
01697
01698 template<>
01699 struct StringMaker<wchar_t const*> {
01700     static std::string convert(wchar_t const* str);
01701 };
01702
01703 template<>
01704 struct StringMaker<wchar_t*> {
01705     static std::string convert(wchar_t* str);
01706 };
01707 #endif
01708
01709 // TBD: Should we use `strlen` to ensure that we don't go out of the buffer,
01710 // while keeping string semantics?
01711 template<int SZ>
01712 struct StringMaker<char[SZ]> {
01713     static std::string convert(char const* str) {
01714         return ::Catch::Detail::stringify(std::string{ str });
01715     }
01716 };
01717
01718 template<int SZ>
01719 struct StringMaker<signed char[SZ]> {

```

```

01718         static std::string convert(signed char const* str) {
01719             return ::Catch::Detail::stringify(std::string{ reinterpret_cast<char const *>(str) });
01720         }
01721     };
01722     template<int SZ>
01723     struct StringMaker<unsigned char[SZ]> {
01724         static std::string convert(unsigned char const* str) {
01725             return ::Catch::Detail::stringify(std::string{ reinterpret_cast<char const *>(str) });
01726         }
01727     };
01728
01729     #if defined(CATCH_CONFIG_CPP17_BYTE)
01730     template<>
01731     struct StringMaker<std::byte> {
01732         static std::string convert(std::byte value);
01733     };
01734     #endif // defined(CATCH_CONFIG_CPP17_BYTE)
01735     template<>
01736     struct StringMaker<int> {
01737         static std::string convert(int value);
01738     };
01739     template<>
01740     struct StringMaker<long> {
01741         static std::string convert(long value);
01742     };
01743     template<>
01744     struct StringMaker<long long> {
01745         static std::string convert(long long value);
01746     };
01747     template<>
01748     struct StringMaker<unsigned int> {
01749         static std::string convert(unsigned int value);
01750     };
01751     template<>
01752     struct StringMaker<unsigned long> {
01753         static std::string convert(unsigned long value);
01754     };
01755     template<>
01756     struct StringMaker<unsigned long long> {
01757         static std::string convert(unsigned long long value);
01758     };
01759
01760     template<>
01761     struct StringMaker<bool> {
01762         static std::string convert(bool b);
01763     };
01764
01765     template<>
01766     struct StringMaker<char> {
01767         static std::string convert(char c);
01768     };
01769     template<>
01770     struct StringMaker<signed char> {
01771         static std::string convert(signed char c);
01772     };
01773     template<>
01774     struct StringMaker<unsigned char> {
01775         static std::string convert(unsigned char c);
01776     };
01777
01778     template<>
01779     struct StringMaker<std::nullptr_t> {
01780         static std::string convert(std::nullptr_t);
01781     };
01782
01783     template<>
01784     struct StringMaker<float> {
01785         static std::string convert(float value);
01786         static int precision;
01787     };
01788
01789     template<>
01790     struct StringMaker<double> {
01791         static std::string convert(double value);
01792         static int precision;
01793     };
01794
01795     template <typename T>
01796     struct StringMaker<T*> {
01797         template <typename U>
01798         static std::string convert(U* p) {
01799             if (p) {
01800                 return ::Catch::Detail::rawMemoryToString(p);
01801             } else {
01802                 return "nullptr";
01803             }
01804         }
01805     };

```

```

01805     };
01806
01807     template <typename R, typename C>
01808     struct StringMaker<R C::*> {
01809         static std::string convert(R C::* p) {
01810             if (p) {
01811                 return ::Catch::Detail::rawMemoryToString(p);
01812             } else {
01813                 return "nullptr";
01814             }
01815         }
01816     };
01817
01818 #if defined(_MANAGED)
01819     template <typename T>
01820     struct StringMaker<T^> {
01821         static std::string convert( T^ ref ) {
01822             return ::Catch::Detail::clrReferenceToString(ref);
01823         }
01824     };
01825 #endif
01826
01827 namespace Detail {
01828     template<typename InputIterator, typename Sentinel = InputIterator>
01829     std::string rangeToString(InputIterator first, Sentinel last) {
01830         ReusableStringStream rss;
01831         rss << "{ ";
01832         if (first != last) {
01833             rss << ::Catch::Detail::stringify(*first);
01834             for (++first; first != last; ++first)
01835                 rss << ", " << ::Catch::Detail::stringify(*first);
01836         }
01837         rss << " }";
01838         return rss.str();
01839     }
01840 }
01841
01842 #ifndef __OBJC__
01843     template<>
01844     struct StringMaker<NSString*> {
01845         static std::string convert(NSString * nsstring) {
01846             if (!nsstring)
01847                 return "nil";
01848             return std::string("@") + [nsstring UTF8String];
01849         }
01850     };
01851     template<>
01852     struct StringMaker<NSObject*> {
01853         static std::string convert(NSObject* nsObject) {
01854             return ::Catch::Detail::stringify([nsObject description]);
01855         }
01856     };
01857
01858     namespace Detail {
01859         inline std::string stringify( NSString* nsstring ) {
01860             return StringMaker<NSString*>::convert( nsstring );
01861         }
01862     }
01863 } // namespace Detail
01864 #endif // __OBJC__
01865
01866 } // namespace Catch
01867
01868 // Separate std-lib types stringification, so it can be selectively enabled
01869 // This means that we do not bring in
01870
01871 #if defined(CATCH_CONFIG_ENABLE_ALL_STRINGMAKERS)
01872 # define CATCH_CONFIG_ENABLE_PAIR_STRINGMAKER
01873 # define CATCH_CONFIG_ENABLE_TUPLE_STRINGMAKER
01874 # define CATCH_CONFIG_ENABLE_VARIANT_STRINGMAKER
01875 # define CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER
01876 # define CATCH_CONFIG_ENABLE_OPTIONAL_STRINGMAKER
01877 #endif
01878
01879 // Separate std::pair specialization
01880 #if defined(CATCH_CONFIG_ENABLE_PAIR_STRINGMAKER)
01881 #include <utility>
01882 namespace Catch {
01883     template<typename T1, typename T2>
01884     struct StringMaker<std::pair<T1, T2> > {
01885         static std::string convert(const std::pair<T1, T2>& pair) {
01886             ReusableStringStream rss;
01887             rss << "{ "
01888                 << ::Catch::Detail::stringify(pair.first)
01889                 << ", "
01890                 << ::Catch::Detail::stringify(pair.second)
01891                 << " }";
01892         }
01893     };

```

```

01893         return rss.str();
01894     }
01895 };
01896 }
01897 #endif // CATCH_CONFIG_ENABLE_PAIR_STRINGMAKER
01898
01899 #if defined(CATCH_CONFIG_ENABLE_OPTIONAL_STRINGMAKER) && defined(CATCH_CONFIG_CPP17_OPTIONAL)
01900 #include <optional>
01901 namespace Catch {
01902     template<typename T>
01903     struct StringMaker<std::optional<T> > {
01904         static std::string convert(const std::optional<T>& optional) {
01905             ReusableStringStream rss;
01906             if (optional.has_value()) {
01907                 rss << ::Catch::Detail::stringify(*optional);
01908             } else {
01909                 rss << "{ }";
01910             }
01911             return rss.str();
01912         }
01913     };
01914 }
01915 #endif // CATCH_CONFIG_ENABLE_OPTIONAL_STRINGMAKER
01916
01917 // Separate std::tuple specialization
01918 #if defined(CATCH_CONFIG_ENABLE_TUPLE_STRINGMAKER)
01919 #include <tuple>
01920 namespace Catch {
01921     namespace Detail {
01922         template<
01923             typename Tuple,
01924             std::size_t N = 0,
01925             bool = (N < std::tuple_size<Tuple>::value)
01926         >
01927         struct TupleElementPrinter {
01928             static void print(const Tuple& tuple, std::ostream& os) {
01929                 os << (N ? ", " : " ")
01930                     << ::Catch::Detail::stringify(std::get<N>(tuple));
01931                 TupleElementPrinter<Tuple, N + 1>::print(tuple, os);
01932             }
01933         };
01934     }
01935
01936     template<
01937         typename Tuple,
01938         std::size_t N
01939     >
01940     struct TupleElementPrinter<Tuple, N, false> {
01941         static void print(const Tuple&, std::ostream&) {}
01942     };
01943 }
01944
01945 template<typename ...Types>
01946 struct StringMaker<std::tuple<Types...> > {
01947     static std::string convert(const std::tuple<Types...>& tuple) {
01948         ReusableStringStream rss;
01949         rss << '{';
01950         Detail::TupleElementPrinter<std::tuple<Types...>::print(tuple, rss.get());
01951         rss << " ";
01952         return rss.str();
01953     }
01954 };
01955 }
01956 #endif // CATCH_CONFIG_ENABLE_TUPLE_STRINGMAKER
01957
01958 #if defined(CATCH_CONFIG_ENABLE_VARIANT_STRINGMAKER) && defined(CATCH_CONFIG_CPP17_VARIANT)
01959 #include <variant>
01960 namespace Catch {
01961     template<>
01962     struct StringMaker<std::monostate> {
01963         static std::string convert(const std::monostate&) {
01964             return "{ }";
01965         }
01966     };
01967
01968     template<typename... Elements>
01969     struct StringMaker<std::variant<Elements...> > {
01970         static std::string convert(const std::variant<Elements...>& variant) {
01971             if (variant.valueless_by_exception()) {
01972                 return "{valueless variant}";
01973             } else {
01974                 return std::visit(
01975                     [](const auto& value) {
01976                         return ::Catch::Detail::stringify(value);
01977                     },
01978                     variant
01979                 );
01980             }
01981         }
01982     };
01983 }
01984 #endif // CATCH_CONFIG_ENABLE_VARIANT_STRINGMAKER

```



```

01980     }
01981     }
01982 };
01983 }
01984 #endif // CATCH_CONFIG_ENABLE_VARIANT_STRINGMAKER
01985
01986 namespace Catch {
01987     // Import begin/ end from std here
01988     using std::begin;
01989     using std::end;
01990
01991     namespace detail {
01992         template <typename...>
01993         struct void_type {
01994             using type = void;
01995         };
01996
01997         template <typename T, typename = void>
01998         struct is_range_impl : std::false_type {
01999         };
02000
02001         template <typename T>
02002         struct is_range_impl<T, typename void_type<decltype(begin(std::declval<T>()))>::type> :
02003             std::true_type {
02004         };
02005     } // namespace detail
02006
02007     template <typename T>
02008     struct is_range : detail::is_range_impl<T> {
02009     };
02010
02011     #if defined(_MANAGED) // Managed types are never ranges
02012     template <typename T>
02013     struct is_range<T^> {
02014         static const bool value = false;
02015     };
02016 #endif
02017
02018     template<typename Range>
02019     std::string rangeToString( Range const& range ) {
02020         return ::Catch::Detail::rangeToString( begin( range ), end( range ) );
02021     }
02022
02023     // Handle vector<bool> specially
02024     template<typename Allocator>
02025     std::string rangeToString( std::vector<bool, Allocator> const& v ) {
02026         ReusableStringStream rss;
02027         rss << "{ ";
02028         bool first = true;
02029         for( bool b : v ) {
02030             if( first )
02031                 first = false;
02032             else
02033                 rss << ", ";
02034             rss << ::Catch::Detail::stringify( b );
02035         }
02036         rss << " }";
02037         return rss.str();
02038     }
02039
02040     template<typename R>
02041     struct StringMaker<R, typename std::enable_if<is_range<R>::value &&
02042         !::Catch::Detail::IsStreamInsertable<R>::value>::type> {
02043         static std::string convert( R const& range ) {
02044             return rangeToString( range );
02045         }
02046     };
02047
02048     template <typename T, int SZ>
02049     struct StringMaker<T[SZ]> {
02050         static std::string convert( T const(&arr)[SZ] ) {
02051             return rangeToString(arr);
02052         }
02053     };
02054
02055 } // namespace Catch
02056
02057 // Separate std::chrono::duration specialization
02058 #if defined(CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER)
02059 #include <ctime>
02060 #include <ratio>
02061 #include <chrono>
02062
02063 namespace Catch {
02064     template <class Ratio>
02065     struct ratio_string {

```

```

02065     static std::string symbol();
02066 };
02067
02068 template <class Ratio>
02069 std::string ratio_string<Ratio>::symbol() {
02070     Catch::ReusableStringStream rss;
02071     rss << '[' << Ratio::num << '/'
02072         << Ratio::den << ']';
02073     return rss.str();
02074 }
02075 template <>
02076 struct ratio_string<std::atto> {
02077     static std::string symbol();
02078 };
02079 template <>
02080 struct ratio_string<std::femto> {
02081     static std::string symbol();
02082 };
02083 template <>
02084 struct ratio_string<std::pico> {
02085     static std::string symbol();
02086 };
02087 template <>
02088 struct ratio_string<std::nano> {
02089     static std::string symbol();
02090 };
02091 template <>
02092 struct ratio_string<std::micro> {
02093     static std::string symbol();
02094 };
02095 template <>
02096 struct ratio_string<std::milli> {
02097     static std::string symbol();
02098 };
02099
02101 // std::chrono::duration specializations
02102 template<typename Value, typename Ratio>
02103 struct StringMaker<std::chrono::duration<Value, Ratio> {
02104     static std::string convert(std::chrono::duration<Value, Ratio> const& duration) {
02105         ReusableStringStream rss;
02106         rss << duration.count() << ' ' << ratio_string<Ratio>::symbol() << 's';
02107         return rss.str();
02108     }
02109 };
02110 template<typename Value>
02111 struct StringMaker<std::chrono::duration<Value, std::ratio<1>> {
02112     static std::string convert(std::chrono::duration<Value, std::ratio<1> const& duration) {
02113         ReusableStringStream rss;
02114         rss << duration.count() << " s";
02115         return rss.str();
02116     }
02117 };
02118 template<typename Value>
02119 struct StringMaker<std::chrono::duration<Value, std::ratio<60>> {
02120     static std::string convert(std::chrono::duration<Value, std::ratio<60> const& duration) {
02121         ReusableStringStream rss;
02122         rss << duration.count() << " m";
02123         return rss.str();
02124     }
02125 };
02126 template<typename Value>
02127 struct StringMaker<std::chrono::duration<Value, std::ratio<3600>> {
02128     static std::string convert(std::chrono::duration<Value, std::ratio<3600> const& duration) {
02129         ReusableStringStream rss;
02130         rss << duration.count() << " h";
02131         return rss.str();
02132     }
02133 };
02134
02136 // std::chrono::time_point specialization
02137 // Generic time_point cannot be specialized, only std::chrono::time_point<system_clock>
02138 template<typename Clock, typename Duration>
02139 struct StringMaker<std::chrono::time_point<Clock, Duration> {
02140     static std::string convert(std::chrono::time_point<Clock, Duration> const& time_point) {
02141         return ::Catch::Detail::stringify(time_point.time_since_epoch()) + " since epoch";
02142     }
02143 };
02144 // std::chrono::time_point<system_clock> specialization
02145 template<typename Duration>
02146 struct StringMaker<std::chrono::time_point<std::chrono::system_clock, Duration> {
02147     static std::string convert(std::chrono::time_point<std::chrono::system_clock, Duration> const&
time_point) {
02148         auto converted = std::chrono::system_clock::to_time_t(time_point);
02149
02150 #ifdef _MSC_VER
02151         std::tm timeInfo = {};
02152         gmtime_s(&timeInfo, &converted);

```

```

02153 #else
02154         std::tm* timeInfo = std::gmtime(&converted);
02155 #endif
02156
02157         auto const timeStampSize = sizeof("2017-01-16T17:06:45Z");
02158         char timeStamp[timeStampSize];
02159         const char * const fmt = "%Y-%m-%dT%H:%M:%SZ";
02160
02161 #ifdef _MSC_VER
02162         std::strftime(timeStamp, timeStampSize, fmt, &timeInfo);
02163 #else
02164         std::strftime(timeStamp, timeStampSize, fmt, timeInfo);
02165 #endif
02166         return std::string(timeStamp);
02167     }
02168 };
02169 }
02170 #endif // CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER
02171
02172 #define INTERNAL_CATCH_REGISTER_ENUM( enumName, ... ) \
02173 namespace Catch { \
02174     template<typename EnumType> struct StringMaker<enumName> { \
02175         static std::string convert( enumName value ) { \
02176             static const auto& enumInfo = \
02177             ::Catch::getMutableRegistryHub().getMutableEnumValuesRegistry().registerEnum( #enumName, #__VA_ARGS__, \
02178             { #__VA_ARGS__ } ); \
02179             return static_cast<std::string>(enumInfo.lookup( static_cast<int>( value ) )); \
02180         } \
02181     }; \
02182 }
02183
02184 #define CATCH_REGISTER_ENUM( enumName, ... ) INTERNAL_CATCH_REGISTER_ENUM( enumName, #__VA_ARGS__ )
02185
02186 #ifdef _MSC_VER
02187 #pragma warning(pop)
02188 #endif
02189 // end catch_tostring.h
02190 #include <iosfwd>
02191
02192 #ifdef _MSC_VER
02193 #pragma warning(push)
02194 #pragma warning(disable:4389) // '==' : signed/unsigned mismatch
02195 #pragma warning(disable:4018) // more "signed/unsigned mismatch"
02196 #pragma warning(disable:4312) // Converting int to T* using reinterpret_cast (issue on x64 platform)
02197 #pragma warning(disable:4180) // qualifier applied to function type has no meaning
02198 #pragma warning(disable:4800) // Forcing result to true or false
02199 #endif
02200 namespace Catch {
02201
02202     struct ITransientExpression {
02203         auto isBinaryExpression() const -> bool { return m_isBinaryExpression; }
02204         auto getResult() const -> bool { return m_result; }
02205         virtual void streamReconstructedExpression( std::ostream &os ) const = 0;
02206
02207         ITransientExpression( bool isBinaryExpression, bool result )
02208             : m_isBinaryExpression( isBinaryExpression ),
02209               m_result( result )
02210         {}
02211
02212         // We don't actually need a virtual destructor, but many static analysers
02213         // complain if it's not here :-|
02214         virtual ~ITransientExpression();
02215
02216         bool m_isBinaryExpression;
02217         bool m_result;
02218     };
02219
02220     void formatReconstructedExpression( std::ostream &os, std::string const& lhs, StringRef op,
02221         std::string const& rhs );
02222
02223     template<typename LhsT, typename RhsT>
02224     class BinaryExpr : public ITransientExpression {
02225     public:
02226         BinaryExpr( LhsT lhs, StringRef op, RhsT rhs )
02227             : m_lhs( lhs ), m_op( op ), m_rhs( rhs )
02228         {}
02229
02230         void streamReconstructedExpression( std::ostream &os ) const override {
02231             formatReconstructedExpression( os, Catch::Detail::stringify( m_lhs ), m_op, Catch::Detail::stringify( m_rhs ) );
02232         }
02233
02234     public:
02235         BinaryExpr( bool comparisonResult, LhsT lhs, StringRef op, RhsT rhs )

```

```

02236         :   ITransientExpression{ true, comparisonResult },
02237         m_lhs( lhs ),
02238         m_op( op ),
02239         m_rhs( rhs )
02240     {}
02241
02242     template<typename T>
02243     auto operator && ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02244         static_assert(always_false<T>::value,
02245             "chained comparisons are not supported inside assertions, "
02246             "wrap the expression inside parentheses, or decompose it");
02247     }
02248
02249     template<typename T>
02250     auto operator || ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02251         static_assert(always_false<T>::value,
02252             "chained comparisons are not supported inside assertions, "
02253             "wrap the expression inside parentheses, or decompose it");
02254     }
02255
02256     template<typename T>
02257     auto operator == ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02258         static_assert(always_false<T>::value,
02259             "chained comparisons are not supported inside assertions, "
02260             "wrap the expression inside parentheses, or decompose it");
02261     }
02262
02263     template<typename T>
02264     auto operator != ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02265         static_assert(always_false<T>::value,
02266             "chained comparisons are not supported inside assertions, "
02267             "wrap the expression inside parentheses, or decompose it");
02268     }
02269
02270     template<typename T>
02271     auto operator > ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02272         static_assert(always_false<T>::value,
02273             "chained comparisons are not supported inside assertions, "
02274             "wrap the expression inside parentheses, or decompose it");
02275     }
02276
02277     template<typename T>
02278     auto operator < ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02279         static_assert(always_false<T>::value,
02280             "chained comparisons are not supported inside assertions, "
02281             "wrap the expression inside parentheses, or decompose it");
02282     }
02283
02284     template<typename T>
02285     auto operator >= ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02286         static_assert(always_false<T>::value,
02287             "chained comparisons are not supported inside assertions, "
02288             "wrap the expression inside parentheses, or decompose it");
02289     }
02290
02291     template<typename T>
02292     auto operator <= ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02293         static_assert(always_false<T>::value,
02294             "chained comparisons are not supported inside assertions, "
02295             "wrap the expression inside parentheses, or decompose it");
02296     }
02297 };
02298
02299 template<typename LhsT>
02300 class UnaryExpr : public ITransientExpression {
02301     LhsT m_lhs;
02302
02303     void streamReconstructedExpression( std::ostream &os ) const override {
02304         os << Catch::Detail::stringify( m_lhs );
02305     }
02306
02307 public:
02308     explicit UnaryExpr( LhsT lhs )
02309         :   ITransientExpression{ false, static_cast<bool>(lhs) },
02310         m_lhs( lhs )
02311     {}
02312 };
02313
02314 // Specialised comparison functions to handle equality comparisons between ints and pointers (NULL
deduces as an int)
02315 template<typename LhsT, typename RhsT>
02316 auto compareEqual( LhsT const& lhs, RhsT const& rhs ) -> bool { return static_cast<bool>(lhs ==
rhs); }
02317 template<typename T>
02318 auto compareEqual( T* const& lhs, int rhs ) -> bool { return lhs == reinterpret_cast<void const*>(
rhs ); }
02319 template<typename T>

```

```

02320     auto compareEqual( T* const& lhs, long rhs ) -> bool { return lhs == reinterpret_cast<void
const*>( rhs ); }
02321     template<typename T>
02322     auto compareEqual( int lhs, T* const& rhs ) -> bool { return reinterpret_cast<void const*>( lhs )
== rhs; }
02323     template<typename T>
02324     auto compareEqual( long lhs, T* const& rhs ) -> bool { return reinterpret_cast<void const*>( lhs )
== rhs; }
02325
02326     template<typename LhsT, typename RhsT>
02327     auto compareNotEqual( LhsT const& lhs, RhsT&& rhs ) -> bool { return static_cast<bool>(lhs !=
rhs); }
02328     template<typename T>
02329     auto compareNotEqual( T* const& lhs, int rhs ) -> bool { return lhs != reinterpret_cast<void
const*>( rhs ); }
02330     template<typename T>
02331     auto compareNotEqual( T* const& lhs, long rhs ) -> bool { return lhs != reinterpret_cast<void
const*>( rhs ); }
02332     template<typename T>
02333     auto compareNotEqual( int lhs, T* const& rhs ) -> bool { return reinterpret_cast<void const*>( lhs
) != rhs; }
02334     template<typename T>
02335     auto compareNotEqual( long lhs, T* const& rhs ) -> bool { return reinterpret_cast<void const*>(
lhs ) != rhs; }
02336
02337     template<typename LhsT>
02338     class ExprLhs {
02339     public:
02340         LhsT m_lhs;
02341         explicit ExprLhs( LhsT lhs ) : m_lhs( lhs ) {}
02342
02343         template<typename RhsT>
02344         auto operator == ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
02345             return { compareEqual( m_lhs, rhs ), m_lhs, "==", rhs };
02346         }
02347         auto operator == ( bool rhs ) -> BinaryExpr<LhsT, bool> const {
02348             return { m_lhs == rhs, m_lhs, "==", rhs };
02349         }
02350
02351         template<typename RhsT>
02352         auto operator != ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
02353             return { compareNotEqual( m_lhs, rhs ), m_lhs, "!", rhs };
02354         }
02355         auto operator != ( bool rhs ) -> BinaryExpr<LhsT, bool> const {
02356             return { m_lhs != rhs, m_lhs, "!", rhs };
02357         }
02358
02359         template<typename RhsT>
02360         auto operator > ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
02361             return { static_cast<bool>(m_lhs > rhs), m_lhs, ">", rhs };
02362         }
02363         template<typename RhsT>
02364         auto operator < ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
02365             return { static_cast<bool>(m_lhs < rhs), m_lhs, "<", rhs };
02366         }
02367         template<typename RhsT>
02368         auto operator >= ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
02369             return { static_cast<bool>(m_lhs >= rhs), m_lhs, ">=", rhs };
02370         }
02371         template<typename RhsT>
02372         auto operator <= ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
02373             return { static_cast<bool>(m_lhs <= rhs), m_lhs, "<=", rhs };
02374         }
02375         template <typename RhsT>
02376         auto operator | (RhsT const& rhs) -> BinaryExpr<LhsT, RhsT const&> const {
02377             return { static_cast<bool>(m_lhs | rhs), m_lhs, "|", rhs };
02378         }
02379         template <typename RhsT>
02380         auto operator & (RhsT const& rhs) -> BinaryExpr<LhsT, RhsT const&> const {
02381             return { static_cast<bool>(m_lhs & rhs), m_lhs, "&", rhs };
02382         }
02383         template <typename RhsT>
02384         auto operator ^ (RhsT const& rhs) -> BinaryExpr<LhsT, RhsT const&> const {
02385             return { static_cast<bool>(m_lhs ^ rhs), m_lhs, "^", rhs };
02386         }
02387
02388         template<typename RhsT>
02389         auto operator && ( RhsT const& ) -> BinaryExpr<LhsT, RhsT const&> const {
02390             static_assert(always_false<RhsT>::value,
02391                 "operator&& is not supported inside assertions, "
02392                 "wrap the expression inside parentheses, or decompose it");
02393         }
02394
02395         template<typename RhsT>
02396         auto operator || ( RhsT const& ) -> BinaryExpr<LhsT, RhsT const&> const {
02397             static_assert(always_false<RhsT>::value,
02398                 "operator|| is not supported inside assertions, "

```

```

02399         "wrap the expression inside parentheses, or decompose it");
02400     }
02401
02402     auto makeUnaryExpr() const -> UnaryExpr<LhsT> {
02403         return UnaryExpr<LhsT>{ m_lhs };
02404     }
02405 };
02406
02407 void handleExpression( ITransientExpression const& expr );
02408
02409 template<typename T>
02410 void handleExpression( ExprLhs<T> const& expr ) {
02411     handleExpression( expr.makeUnaryExpr() );
02412 }
02413
02414 struct Decomposer {
02415     template<typename T>
02416     auto operator <= ( T const& lhs ) -> ExprLhs<T const&> {
02417         return ExprLhs<T const&>{ lhs };
02418     }
02419
02420     auto operator <=( bool value ) -> ExprLhs<bool> {
02421         return ExprLhs<bool>{ value };
02422     }
02423 };
02424
02425 } // end namespace Catch
02426
02427 #ifdef _MSC_VER
02428 #pragma warning(pop)
02429 #endif
02430
02431 // end catch_decomposer.h
02432 // start catch_interfaces_capture.h
02433
02434 #include <string>
02435 #include <chrono>
02436
02437 namespace Catch {
02438
02439     class AssertionResult;
02440     struct AssertionInfo;
02441     struct SectionInfo;
02442     struct SectionEndInfo;
02443     struct MessageInfo;
02444     struct MessageBuilder;
02445     struct Counts;
02446     struct AssertionReaction;
02447     struct SourceLineInfo;
02448
02449     struct ITransientExpression;
02450     struct IGeneratorTracker;
02451
02452     #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
02453     struct BenchmarkInfo;
02454     template <typename Duration = std::chrono::duration<double, std::nano>>
02455     struct BenchmarkStats;
02456     #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
02457
02458     struct IResultCapture {
02459
02460         virtual ~IResultCapture();
02461
02462         virtual bool sectionStarted(     SectionInfo const& sectionInfo,
02463                                     Counts& assertions ) = 0;
02464         virtual void sectionEnded( SectionEndInfo const& endInfo ) = 0;
02465         virtual void sectionEndedEarly( SectionEndInfo const& endInfo ) = 0;
02466
02467         virtual auto acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo
02468     ) -> IGeneratorTracker& = 0;
02469
02470     #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
02471         virtual void benchmarkPreparing( std::string const& name ) = 0;
02472         virtual void benchmarkStarting( BenchmarkInfo const& info ) = 0;
02473         virtual void benchmarkEnded( BenchmarkStats<> const& stats ) = 0;
02474         virtual void benchmarkFailed( std::string const& error ) = 0;
02475     #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
02476
02477         virtual void pushScopedMessage( MessageInfo const& message ) = 0;
02478         virtual void popScopedMessage( MessageInfo const& message ) = 0;
02479
02480         virtual void emplaceUnscopedMessage( MessageBuilder const& builder ) = 0;
02481
02482         virtual void handleFatalErrorCondition( StringRef message ) = 0;
02483
02484         virtual void handleExpr
02485             ( AssertionInfo const& info,

```

```

02485         ITransientExpression const& expr,
02486         AssertionReaction& reaction ) = 0;
02487     virtual void handleMessage
02488     (     AssertionInfo const& info,
02489         ResultWas::OfType resultType,
02490         StringRef const& message,
02491         AssertionReaction& reaction ) = 0;
02492     virtual void handleUnexpectedExceptionNotThrown
02493     (     AssertionInfo const& info,
02494         AssertionReaction& reaction ) = 0;
02495     virtual void handleUnexpectedInflightException
02496     (     AssertionInfo const& info,
02497         std::string const& message,
02498         AssertionReaction& reaction ) = 0;
02499     virtual void handleIncomplete
02500     (     AssertionInfo const& info ) = 0;
02501     virtual void handleNonExpr
02502     (     AssertionInfo const& info,
02503         ResultWas::OfType resultType,
02504         AssertionReaction& reaction ) = 0;
02505
02506     virtual bool lastAssertionPassed() = 0;
02507     virtual void assertionPassed() = 0;
02508
02509     // Deprecated, do not use:
02510     virtual std::string getCurrentTestName() const = 0;
02511     virtual const AssertionResult* getLastResult() const = 0;
02512     virtual void exceptionEarlyReported() = 0;
02513 };
02514
02515 IResultCapture& getResultCapture();
02516 }
02517
02518 // end catch_interfaces_capture.h
02519 namespace Catch {
02520
02521     struct TestFailureException{};
02522     struct AssertionResultData;
02523     struct IResultCapture;
02524     class RunContext;
02525
02526     class LazyExpression {
02527     friend class AssertionHandler;
02528     friend struct AssertionStats;
02529     friend class RunContext;
02530
02531     ITransientExpression const* m_transientExpression = nullptr;
02532     bool m_isNegated;
02533     public:
02534         LazyExpression( bool isNegated );
02535         LazyExpression( LazyExpression const& other );
02536         LazyExpression& operator = ( LazyExpression const& ) = delete;
02537
02538         explicit operator bool() const;
02539
02540         friend auto operator << ( std::ostream& os, LazyExpression const& lazyExpr ) -> std::ostream&;
02541     };
02542
02543     struct AssertionReaction {
02544         bool shouldDebugBreak = false;
02545         bool shouldThrow = false;
02546     };
02547
02548     class AssertionHandler {
02549     public:
02550         AssertionInfo m_assertionInfo;
02551         AssertionReaction m_reaction;
02552         bool m_completed = false;
02553         IResultCapture& m_resultCapture;
02554
02555         AssertionHandler
02556         (     StringRef const& macroName,
02557             SourceLineInfo const& lineInfo,
02558             StringRef capturedExpression,
02559             ResultDisposition::Flags resultDisposition );
02560         ~AssertionHandler() {
02561             if ( !m_completed ) {
02562                 m_resultCapture.handleIncomplete( m_assertionInfo );
02563             }
02564         }
02565
02566         template<typename T>
02567         void handleExpr( ExprLhs<T> const& expr ) {
02568             handleExpr( expr.makeUnaryExpr() );
02569         }
02570         void handleExpr( ITransientExpression const& expr );
02571

```

```

02572         void handleMessage(ResultWas::OfType resultType, StringRef const& message);
02573
02574         void handleExceptionThrownAsExpected();
02575         void handleUnexpectedExceptionNotThrown();
02576         void handleExceptionNotThrownAsExpected();
02577         void handleThrowingCallSkipped();
02578         void handleUnexpectedInflightException();
02579
02580         void complete();
02581         void setCompleted();
02582
02583         // query
02584         auto allowThrows() const -> bool;
02585     };
02586
02587     void handleExceptionMatchExpr( AssertionHandler& handler, std::string const& str, StringRef const&
matcherString );
02588
02589 } // namespace Catch
02590
02591 // end catch_assertionhandler.h
02592 // start catch_message.h
02593
02594 #include <string>
02595 #include <vector>
02596
02597 namespace Catch {
02598
02599     struct MessageInfo {
02600         MessageInfo( StringRef const& _macroName,
02601                     SourceLineInfo const& _lineInfo,
02602                     ResultWas::OfType _type );
02603
02604         StringRef macroName;
02605         std::string message;
02606         SourceLineInfo lineInfo;
02607         ResultWas::OfType type;
02608         unsigned int sequence;
02609
02610         bool operator == ( MessageInfo const& other ) const;
02611         bool operator < ( MessageInfo const& other ) const;
02612     private:
02613         static unsigned int globalCount;
02614     };
02615
02616     struct MessageStream {
02617
02618         template<typename T>
02619         MessageStream& operator << ( T const& value ) {
02620             m_stream << value;
02621             return *this;
02622         }
02623
02624         ReusableStringStream m_stream;
02625     };
02626
02627     struct MessageBuilder : MessageStream {
02628         MessageBuilder( StringRef const& macroName,
02629                        SourceLineInfo const& lineInfo,
02630                        ResultWas::OfType type );
02631
02632         template<typename T>
02633         MessageBuilder& operator <<( T const& value ) {
02634             m_stream << value;
02635             return *this;
02636         }
02637
02638         MessageInfo m_info;
02639     };
02640
02641     class ScopedMessage {
02642     public:
02643         explicit ScopedMessage( MessageBuilder const& builder );
02644         ScopedMessage( ScopedMessage& duplicate ) = delete;
02645         ScopedMessage( ScopedMessage&& old );
02646         ~ScopedMessage();
02647
02648         MessageInfo m_info;
02649         bool m_moved;
02650     };
02651
02652     class Capturer {
02653     public:
02654         std::vector<MessageInfo> m_messages;
02655         IResultCapture& m_resultCapture = getResultCapture();
02656         size_t m_captured = 0;
02657         Capturer( StringRef macroName, SourceLineInfo const& lineInfo, ResultWas::OfType resultType,

```



```

StringRef names );
02658     ~Capturer();
02659
02660     void captureValue( size_t index, std::string const& value );
02661
02662     template<typename T>
02663     void captureValues( size_t index, T const& value ) {
02664         captureValue( index, Catch::Detail::stringify( value ) );
02665     }
02666
02667     template<typename T, typename... Ts>
02668     void captureValues( size_t index, T const& value, Ts const&... values ) {
02669         captureValue( index, Catch::Detail::stringify(value) );
02670         captureValues( index+1, values... );
02671     }
02672 };
02673
02674 } // end namespace Catch
02675
02676 // end catch_message.h
02677 #if !defined(CATCH_CONFIG_DISABLE)
02678
02679 #if !defined(CATCH_CONFIG_DISABLE_STRINGIFICATION)
02680     #define CATCH_INTERNAL_STRINGIFY(...) #__VA_ARGS__
02681 #else
02682     #define CATCH_INTERNAL_STRINGIFY(...) "Disabled by CATCH_CONFIG_DISABLE_STRINGIFICATION"
02683 #endif
02684
02685 #if defined(CATCH_CONFIG_FAST_COMPILE) || defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
02686
02687 // Another way to speed-up compilation is to omit local try-catch for REQUIRE*
02688 // macros.
02689 #define INTERNAL_CATCH_TRY
02690 #define INTERNAL_CATCH_CATCH( handler )
02691
02692 #else // CATCH_CONFIG_FAST_COMPILE
02693 #define INTERNAL_CATCH_TRY try
02694 #define INTERNAL_CATCH_CATCH( handler ) catch(...) { handler.handleUnexpectedInflightException(); }
02695 #endif
02696
02697 #define INTERNAL_CATCH_REACT( handler ) handler.complete();
02698
02699 #define INTERNAL_CATCH_TEST( macroName, resultDisposition, ... ) \
02700     do { \
02701         CATCH_INTERNAL_IGNORE_BUT_WARN(__VA_ARGS__); \
02702         Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, \
02703             CATCH_INTERNAL_STRINGIFY(__VA_ARGS__), resultDisposition ); \
02704         INTERNAL_CATCH_TRY { \
02705             CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
02706             CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS \
02707             catchAssertionHandler.handleExpr( Catch::Decomposer() <= __VA_ARGS__ ); \
02708             CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
02709         } INTERNAL_CATCH_CATCH( catchAssertionHandler ) \
02710         INTERNAL_CATCH_REACT( catchAssertionHandler ) \
02711     } while( (void)0, (false) && static_cast<bool>( !!( __VA_ARGS__ ) ) )
02712
02713 #define INTERNAL_CATCH_IF( macroName, resultDisposition, ... ) \
02714     INTERNAL_CATCH_TEST( macroName, resultDisposition, __VA_ARGS__ ); \
02715     if( Catch::getResultCapture().lastAssertionPassed() )
02716
02717 #define INTERNAL_CATCH_ELSE( macroName, resultDisposition, ... ) \
02718     INTERNAL_CATCH_TEST( macroName, resultDisposition, __VA_ARGS__ ); \
02719     if( !Catch::getResultCapture().lastAssertionPassed() )
02720
02721 #define INTERNAL_CATCH_NO_THROW( macroName, resultDisposition, ... ) \
02722     do { \
02723         Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, \
02724             CATCH_INTERNAL_STRINGIFY(__VA_ARGS__), resultDisposition ); \
02725         try { \
02726             static_cast<void>(__VA_ARGS__); \
02727             catchAssertionHandler.handleExceptionNotThrownAsExpected(); \
02728         } \
02729         catch( ... ) { \
02730             catchAssertionHandler.handleUnexpectedInflightException(); \
02731         } \
02732         INTERNAL_CATCH_REACT( catchAssertionHandler ) \
02733     } while( false )
02734
02735 #define INTERNAL_CATCH_THROWS( macroName, resultDisposition, ... ) \
02736     do { \
02737         Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, \
02738             CATCH_INTERNAL_STRINGIFY(__VA_ARGS__), resultDisposition ); \
02739         if( catchAssertionHandler.allowThrows() ) \
02740             try { \
02741                 static_cast<void>(__VA_ARGS__); \
02742

```

```

02747         catchAssertionHandler.handleUnexpectedExceptionNotThrown(); \
02748     } \
02749     catch( ... ) { \
02750         catchAssertionHandler.handleExceptionThrownAsExpected(); \
02751     } \
02752     else \
02753         catchAssertionHandler.handleThrowingCallSkipped(); \
02754     INTERNAL_CATCH_REACT( catchAssertionHandler ) \
02755 } while( false )
02756
02757 #define INTERNAL_CATCH_THROWS_AS( macroName, exceptionType, resultDisposition, expr ) \
02758 do { \
02759     Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, \
02760     CATCH_INTERNAL_STRINGIFY(expr) ", " CATCH_INTERNAL_STRINGIFY(exceptionType), resultDisposition ); \
02761     if( catchAssertionHandler.allowThrows() ) \
02762         try { \
02763             static_cast<void>(expr); \
02764             catchAssertionHandler.handleUnexpectedExceptionNotThrown(); \
02765         } \
02766         catch( exceptionType const& ) { \
02767             catchAssertionHandler.handleExceptionThrownAsExpected(); \
02768         } \
02769         catch( ... ) { \
02770             catchAssertionHandler.handleUnexpectedInflightException(); \
02771         } \
02772     else \
02773         catchAssertionHandler.handleThrowingCallSkipped(); \
02774     INTERNAL_CATCH_REACT( catchAssertionHandler ) \
02775 } while( false )
02776
02777 #define INTERNAL_CATCH_MSG( macroName, messageType, resultDisposition, ... ) \
02778 do { \
02779     Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, \
02780     Catch::StringRef(), resultDisposition ); \
02781     catchAssertionHandler.handleMessage( messageType, ( Catch::MessageStream() << __VA_ARGS__ + \
02782     ::Catch::StreamEndStop() ).m_stream.str() ); \
02783     INTERNAL_CATCH_REACT( catchAssertionHandler ) \
02784 } while( false )
02785
02786 #define INTERNAL_CATCH_CAPTURE( varName, macroName, ... ) \
02787 auto varName = Catch::Capturer( macroName, CATCH_INTERNAL_LINEINFO, Catch::ResultWas::Info, \
02788 __VA_ARGS__ ); \
02789 varName.captureValues( 0, __VA_ARGS__ )
02790
02791 #define INTERNAL_CATCH_INFO( macroName, log ) \
02792 Catch::ScopedMessage INTERNAL_CATCH_UNIQUE_NAME( scopedMessage )( Catch::MessageBuilder( \
02793 macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, Catch::ResultWas::Info ) << log );
02794
02795 #define INTERNAL_CATCH_UNSCOPED_INFO( macroName, log ) \
02796 Catch::getResultCapture().emplaceUnscopedMessage( Catch::MessageBuilder( macroName##_catch_sr, \
02797 CATCH_INTERNAL_LINEINFO, Catch::ResultWas::Info ) << log )
02798
02799 // Although this is matcher-based, it can be used with just a string
02800 #define INTERNAL_CATCH_THROWS_STR_MATCHES( macroName, resultDisposition, matcher, ... ) \
02801 do { \
02802     Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, \
02803     CATCH_INTERNAL_STRINGIFY(__VA_ARGS__) ", " CATCH_INTERNAL_STRINGIFY(matcher), resultDisposition ); \
02804     if( catchAssertionHandler.allowThrows() ) \
02805         try { \
02806             static_cast<void>(__VA_ARGS__); \
02807             catchAssertionHandler.handleUnexpectedExceptionNotThrown(); \
02808         } \
02809         catch( ... ) { \
02810             Catch::handleExceptionMatchExpr( catchAssertionHandler, matcher, #matcher##_catch_sr \
02811 ); \
02812         } \
02813     else \
02814         catchAssertionHandler.handleThrowingCallSkipped(); \
02815     INTERNAL_CATCH_REACT( catchAssertionHandler ) \
02816 } while( false )
02817
02818 #endif // CATCH_CONFIG_DISABLE
02819
02820 // end catch_capture.hpp
02821 // start catch_section.h
02822 // start catch_section_info.h
02823 // start catch_totals.h
02824
02825 #include <cstdlib>
02826
02827 namespace Catch {
02828
02829     struct Counts {
02830         Counts operator - ( Counts const& other ) const;
02831         Counts& operator += ( Counts const& other );

```

```

02832
02833     std::size_t total() const;
02834     bool allPassed() const;
02835     bool allOk() const;
02836
02837     std::size_t passed = 0;
02838     std::size_t failed = 0;
02839     std::size_t failedButOk = 0;
02840 };
02841
02842 struct Totals {
02843
02844     Totals operator - ( Totals const& other ) const;
02845     Totals& operator += ( Totals const& other );
02846
02847     Totals delta( Totals const& prevTotals ) const;
02848
02849     int error = 0;
02850     Counts assertions;
02851     Counts testCases;
02852 };
02853 }
02854
02855 // end catch_totals.h
02856 #include <string>
02857 namespace Catch {
02858
02859     struct SectionInfo {
02860         SectionInfo
02861             ( SourceLineInfo const& _lineInfo,
02862               std::string const& _name );
02863
02864         // Deprecated
02865         SectionInfo
02866             ( SourceLineInfo const& _lineInfo,
02867               std::string const& _name,
02868               std::string const& ) : SectionInfo( _lineInfo, _name ) {}
02869
02870         std::string name;
02871         std::string description; // !Deprecated: this will always be empty
02872         SourceLineInfo lineInfo;
02873     };
02874
02875     struct SectionEndInfo {
02876         SectionInfo sectionInfo;
02877         Counts prevAssertions;
02878         double durationInSeconds;
02879     };
02880 };
02881 } // end namespace Catch
02882
02883 // end catch_section_info.h
02884 // start catch_timer.h
02885 #include <cstdint>
02886 namespace Catch {
02887
02888     auto getCurrentNanosecondsSinceEpoch() -> uint64_t;
02889     auto getEstimatedClockResolution() -> uint64_t;
02890
02891     class Timer {
02892     public:
02893         uint64_t m_nanoseconds = 0;
02894         void start();
02895         auto getElapsedNanoseconds() const -> uint64_t;
02896         auto getElapsedMicroseconds() const -> uint64_t;
02897         auto getElapsedMilliseconds() const -> unsigned int;
02898         auto getElapsedSeconds() const -> double;
02899     };
02900
02901 } // namespace Catch
02902
02903 // end catch_timer.h
02904 #include <string>
02905 namespace Catch {
02906
02907     class Section : NonCopyable {
02908     public:
02909         Section( SectionInfo const& info );
02910         ~Section();
02911
02912         // This indicates whether the section should be executed or not
02913         explicit operator bool() const;
02914     };
02915 }

```

```

02919     private:
02920         SectionInfo m_info;
02921
02922         std::string m_name;
02923         Counts m_assertions;
02924         bool m_sectionIncluded;
02925         Timer m_timer;
02926     };
02927
02928 } // end namespace Catch
02929
02930 #define INTERNAL_CATCH_SECTION( ... ) \
02931     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
02932     CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS \
02933     if( Catch::Section const& INTERNAL_CATCH_UNIQUE_NAME( catch_internal_Section ) = \
02934         Catch::SectionInfo( CATCH_INTERNAL_LINEINFO, __VA_ARGS__ ) ) \
02935         CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
02936
02937 #define INTERNAL_CATCH_DYNAMIC_SECTION( ... ) \
02938     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
02939     CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS \
02940     if( Catch::Section const& INTERNAL_CATCH_UNIQUE_NAME( catch_internal_Section ) = \
02941         Catch::SectionInfo( CATCH_INTERNAL_LINEINFO, (Catch::ReusableStringStream() << __VA_ARGS__).str() ) ) \
02942         CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
02943
02944 // end catch_section.h
02945 // start catch_interfaces_exception.h
02946
02947 // start catch_interfaces_registry_hub.h
02948
02949 #include <string>
02950 #include <memory>
02951
02952 namespace Catch {
02953
02954     class TestCase;
02955     struct ITestCaseRegistry;
02956     struct IExceptionTranslatorRegistry;
02957     struct IExceptionTranslator;
02958     struct IReporterRegistry;
02959     struct IReporterFactory;
02960     struct ITagAliasRegistry;
02961     struct IMutableEnumValuesRegistry;
02962
02963     class StartupExceptionRegistry;
02964
02965     using IReporterFactoryPtr = std::shared_ptr<IReporterFactory>;
02966
02967     struct IRegistryHub {
02968     public:
02969         virtual ~IRegistryHub();
02970
02971         virtual IReporterRegistry const& getReporterRegistry() const = 0;
02972         virtual ITestCaseRegistry const& getTestCaseRegistry() const = 0;
02973         virtual ITagAliasRegistry const& getTagAliasRegistry() const = 0;
02974         virtual IExceptionTranslatorRegistry const& getExceptionTranslatorRegistry() const = 0;
02975         virtual StartupExceptionRegistry const& getStartupExceptionRegistry() const = 0;
02976     };
02977
02978     struct IMutableRegistryHub {
02979     public:
02980         virtual ~IMutableRegistryHub();
02981         virtual void registerReporter( std::string const& name, IReporterFactoryPtr const& factory ) = 0;
02982         virtual void registerListener( IReporterFactoryPtr const& factory ) = 0;
02983         virtual void registerTest( TestCase const& testInfo ) = 0;
02984         virtual void registerTranslator( const IExceptionTranslator* translator ) = 0;
02985         virtual void registerTagAlias( std::string const& alias, std::string const& tag, SourceLineInfo const& lineInfo ) = 0;
02986         virtual void registerStartupException() noexcept = 0;
02987         virtual IMutableEnumValuesRegistry& getMutableEnumValuesRegistry() = 0;
02988     };
02989
02990     IRegistryHub const& getRegistryHub();
02991     IMutableRegistryHub& getMutableRegistryHub();
02992     void cleanUp();
02993     std::string translateActiveException();
02994 }
02995
02996 // end catch_interfaces_registry_hub.h
02997
02998 #if defined(CATCH_CONFIG_DISABLE)
02999     #define INTERNAL_CATCH_TRANSLATE_EXCEPTION_NO_REG( translatorName, signature ) \
03000     static std::string translatorName( signature )
03001
03002 #endif
03003
03004 #include <exception>
03005 #include <string>

```

```

03002 #include <vector>
03003
03004 namespace Catch {
03005     using exceptionTranslateFunction = std::string(*)();
03006
03007     struct IExceptionTranslator;
03008     using ExceptionTranslators = std::vector<std::unique_ptr<IExceptionTranslator const>;
03009
03010     struct IExceptionTranslator {
03011         virtual ~IExceptionTranslator();
03012         virtual std::string translate( ExceptionTranslators::const_iterator it,
ExceptionTranslators::const_iterator itEnd ) const = 0;
03013     };
03014
03015     struct IExceptionTranslatorRegistry {
03016         virtual ~IExceptionTranslatorRegistry();
03017
03018         virtual std::string translateActiveException() const = 0;
03019     };
03020
03021     class ExceptionTranslatorRegistrar {
03022     public:
03023         template<typename T>
03024         class ExceptionTranslator : public IExceptionTranslator {
03025         public:
03026             ExceptionTranslator( std::string(*translateFunction)( T& ) )
03027             : m_translateFunction( translateFunction )
03028             {}
03029
03030             std::string translate( ExceptionTranslators::const_iterator it,
ExceptionTranslators::const_iterator itEnd ) const override {
03031             #if defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
03032                 return "";
03033             #else
03034                 try {
03035                     if( it == itEnd )
03036                         std::rethrow_exception(std::current_exception());
03037                     else
03038                         return (*it)->translate( it+1, itEnd );
03039                 }
03040                 catch( T& ex ) {
03041                     return m_translateFunction( ex );
03042                 }
03043             #endif
03044             }
03045
03046         protected:
03047             std::string(*m_translateFunction)( T& );
03048         };
03049
03050     public:
03051         template<typename T>
03052         ExceptionTranslatorRegistrar( std::string(*translateFunction)( T& ) ) {
03053             getMutableRegistryHub().registerTranslator
03054                 ( new ExceptionTranslator<T>( translateFunction ) );
03055         }
03056     };
03057 }
03058
03059 #define INTERNAL_CATCH_TRANSLATE_EXCEPTION2( translatorName, signature ) \
03060     static std::string translatorName( signature ); \
03061     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
03062     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
03063     namespace{ Catch::ExceptionTranslatorRegistrar INTERNAL_CATCH_UNIQUE_NAME( \
catch_internal_ExceptionTranslator )( &translatorName ); } \
03064     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
03065     static std::string translatorName( signature )
03066
03067 #define INTERNAL_CATCH_TRANSLATE_EXCEPTION( signature ) INTERNAL_CATCH_TRANSLATE_EXCEPTION2( \
INTERNAL_CATCH_UNIQUE_NAME( catch_internal_ExceptionTranslator ), signature )
03068
03069 // end catch_interfaces_exception.h
03070 // start catch_approx.h
03071 #include <type_traits>
03072
03073 namespace Catch {
03074 namespace Detail {
03075     class Approx {
03076     private:
03077         bool equalityComparisonImpl(double other) const;
03078         // Validates the new margin (margin >= 0)
03079         // out-of-line to avoid including stdexcept in the header
03080         void setMargin(double margin);
03081         // Validates the new epsilon (0 < epsilon < 1)
03082         // out-of-line to avoid including stdexcept in the header

```

```

03086         void setEpsilon(double epsilon);
03087
03088     public:
03089         explicit Approx ( double value );
03090
03091         static Approx custom();
03092
03093         Approx operator-( ) const;
03094
03095
03096     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03097         Approx operator()( T const& value ) const {
03098         Approx approx( static_cast<double>(value) );
03099         approx.m_epsilon = m_epsilon;
03100         approx.m_margin = m_margin;
03101         approx.m_scale = m_scale;
03102         return approx;
03103     }
03104
03105     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03106     explicit Approx( T const& value ): Approx(static_cast<double>(value))
03107     {}
03108
03109     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03110     friend bool operator == ( const T& lhs, Approx const& rhs ) {
03111         auto lhs_v = static_cast<double>(lhs);
03112         return rhs.equalityComparisonImpl(lhs_v);
03113     }
03114
03115     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03116     friend bool operator == ( Approx const& lhs, const T& rhs ) {
03117         return operator==( rhs, lhs );
03118     }
03119
03120     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03121     friend bool operator != ( T const& lhs, Approx const& rhs ) {
03122         return !operator==( lhs, rhs );
03123     }
03124
03125     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03126     friend bool operator != ( Approx const& lhs, T const& rhs ) {
03127         return !operator==( rhs, lhs );
03128     }
03129
03130     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03131     friend bool operator <= ( T const& lhs, Approx const& rhs ) {
03132         return static_cast<double>(lhs) < rhs.m_value || lhs == rhs;
03133     }
03134
03135     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03136     friend bool operator <= ( Approx const& lhs, T const& rhs ) {
03137         return lhs.m_value < static_cast<double>(rhs) || lhs == rhs;
03138     }
03139
03140     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03141     friend bool operator >= ( T const& lhs, Approx const& rhs ) {
03142         return static_cast<double>(lhs) > rhs.m_value || lhs == rhs;
03143     }
03144
03145     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03146     friend bool operator >= ( Approx const& lhs, T const& rhs ) {
03147         return lhs.m_value > static_cast<double>(rhs) || lhs == rhs;
03148     }
03149
03150     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03151     Approx& epsilon( T const& newEpsilon ) {
03152         double epsilonAsDouble = static_cast<double>(newEpsilon);
03153         setEpsilon(epsilonAsDouble);
03154         return *this;
03155     }
03156
03157     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03158     Approx& margin( T const& newMargin ) {
03159         double marginAsDouble = static_cast<double>(newMargin);
03160         setMargin(marginAsDouble);
03161         return *this;

```

```

03161     }
03162
03163     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03164         Approx& scale( T const& newScale ) {
03165         m_scale = static_cast<double>(newScale);
03166         return *this;
03167     }
03168
03169     std::string toString() const;
03170
03171     private:
03172         double m_epsilon;
03173         double m_margin;
03174         double m_scale;
03175         double m_value;
03176     };
03177 } // end namespace Detail
03178
03179 namespace literals {
03180     Detail::Approx operator "" _a(long double val);
03181     Detail::Approx operator "" _a(unsigned long long val);
03182 } // end namespace literals
03183
03184 template<>
03185 struct StringMaker<Catch::Detail::Approx> {
03186     static std::string convert(Catch::Detail::Approx const& value);
03187 };
03188
03189 } // end namespace Catch
03190
03191 // end catch_approx.h
03192 // start catch_string_manip.h
03193
03194 #include <string>
03195 #include <iosfwd>
03196 #include <vector>
03197
03198 namespace Catch {
03199
03200     bool startsWith( std::string const& s, std::string const& prefix );
03201     bool startsWith( std::string const& s, char prefix );
03202     bool endsWith( std::string const& s, std::string const& suffix );
03203     bool endsWith( std::string const& s, char suffix );
03204     bool contains( std::string const& s, std::string const& infix );
03205     void toLowerInPlace( std::string& s );
03206     std::string toLower( std::string const& s );
03207     std::string trim( std::string const& str );
03208     StringRef trim(StringRef ref);
03209
03210     // !!! Be aware, returns refs into original string - make sure original string outlives them
03211     std::vector<StringRef> splitStringRef( StringRef str, char delimiter );
03212     bool replaceInPlace( std::string& str, std::string const& replaceThis, std::string const& withThis );
03213
03214     struct pluralise {
03215         pluralise( std::size_t count, std::string const& label );
03216
03217         friend std::ostream& operator << ( std::ostream& os, pluralise const& pluraliser );
03218
03219         std::size_t m_count;
03220         std::string m_label;
03221     };
03222 }
03223
03224 // end catch_string_manip.h
03225 #ifndef CATCH_CONFIG_DISABLE_MATCHERS
03226 // start catch_capture_matchers.h
03227
03228 // start catch_matchers.h
03229
03230 #include <string>
03231 #include <vector>
03232
03233 namespace Catch {
03234     namespace Matchers {
03235         namespace Impl {
03236
03237             template<typename ArgT> struct MatchAllOf;
03238             template<typename ArgT> struct MatchAnyOf;
03239             template<typename ArgT> struct MatchNotOf;
03240
03241             class MatcherUntypedBase {
03242             public:
03243                 MatcherUntypedBase() = default;
03244                 MatcherUntypedBase ( MatcherUntypedBase const& ) = default;
03245                 MatcherUntypedBase& operator = ( MatcherUntypedBase const& ) = delete;

```

```

03248         std::string toString() const;
03249
03250     protected:
03251         virtual ~MatcherUntypedBase();
03252         virtual std::string describe() const = 0;
03253         mutable std::string m_cachedToString;
03254     };
03255
03256 #ifndef __clang__
03257 #     pragma clang diagnostic push
03258 #     pragma clang diagnostic ignored "-Wnon-virtual-dtor"
03259 #endif
03260
03261     template<typename ObjectT>
03262     struct MatcherMethod {
03263         virtual bool match( ObjectT const& arg ) const = 0;
03264     };
03265
03266 #if defined(__OBJC__)
03267     // Hack to fix Catch GH issue #1661. Could use id for generic Object support.
03268     // use of const for Object pointers is very uncommon and under ARC it causes some kind of
    signature mismatch that breaks compilation
03269     template<>
03270     struct MatcherMethod<NSString*> {
03271         virtual bool match( NSString* arg ) const = 0;
03272     };
03273 #endif
03274
03275 #ifndef __clang__
03276 #     pragma clang diagnostic pop
03277 #endif
03278
03279     template<typename T>
03280     struct MatcherBase : MatcherUntypedBase, MatcherMethod<T> {
03281
03282         MatchAllOf<T> operator && ( MatcherBase const& other ) const;
03283         MatchAnyOf<T> operator || ( MatcherBase const& other ) const;
03284         MatchNotOf<T> operator ! ( ) const;
03285     };
03286
03287     template<typename ArgT>
03288     struct MatchAllOf : MatcherBase<ArgT> {
03289         bool match( ArgT const& arg ) const override {
03290             for( auto matcher : m_matchers ) {
03291                 if (!matcher->match(arg))
03292                     return false;
03293             }
03294             return true;
03295         }
03296         std::string describe() const override {
03297             std::string description;
03298             description.reserve( 4 + m_matchers.size()*32 );
03299             description += "( ";
03300             bool first = true;
03301             for( auto matcher : m_matchers ) {
03302                 if( first )
03303                     first = false;
03304                 else
03305                     description += " and ";
03306                 description += matcher->toString();
03307             }
03308             description += " )";
03309             return description;
03310         }
03311
03312         MatchAllOf<ArgT> operator && ( MatcherBase<ArgT> const& other ) {
03313             auto copy(*this);
03314             copy.m_matchers.push_back( &other );
03315             return copy;
03316         }
03317
03318         std::vector<MatcherBase<ArgT> const*> m_matchers;
03319     };
03320
03321     template<typename ArgT>
03322     struct MatchAnyOf : MatcherBase<ArgT> {
03323
03324         bool match( ArgT const& arg ) const override {
03325             for( auto matcher : m_matchers ) {
03326                 if (matcher->match(arg))
03327                     return true;
03328             }
03329             return false;
03330         }
03331         std::string describe() const override {
03332             std::string description;
03333             description.reserve( 4 + m_matchers.size()*32 );
03334             description += "( ";

```



```

03334         bool first = true;
03335         for( auto matcher : m_matchers ) {
03336             if( first )
03337                 first = false;
03338             else
03339                 description += " or ";
03340             description += matcher->toString();
03341         }
03342         description += " )";
03343         return description;
03344     }
03345
03346     MatchAnyOf<ArgT> operator || ( MatcherBase<ArgT> const& other ) {
03347         auto copy(*this);
03348         copy.m_matchers.push_back( &other );
03349         return copy;
03350     }
03351
03352     std::vector<MatcherBase<ArgT> const*> m_matchers;
03353 };
03354
03355 template<typename ArgT>
03356 struct MatchNotOf : MatcherBase<ArgT> {
03357
03358     MatchNotOf( MatcherBase<ArgT> const& underlyingMatcher ) : m_underlyingMatcher(
underlyingMatcher ) {}
03359
03360     bool match( ArgT const& arg ) const override {
03361         return !m_underlyingMatcher.match( arg );
03362     }
03363
03364     std::string describe() const override {
03365         return "not " + m_underlyingMatcher.toString();
03366     }
03367     MatcherBase<ArgT> const& m_underlyingMatcher;
03368 };
03369
03370 template<typename T>
03371 MatchAllOf<T> MatcherBase<T>::operator && ( MatcherBase const& other ) const {
03372     return MatchAllOf<T>() && *this && other;
03373 }
03374 template<typename T>
03375 MatchAnyOf<T> MatcherBase<T>::operator || ( MatcherBase const& other ) const {
03376     return MatchAnyOf<T>() || *this || other;
03377 }
03378 template<typename T>
03379 MatchNotOf<T> MatcherBase<T>::operator ! () const {
03380     return MatchNotOf<T>( *this );
03381 }
03382
03383 } // namespace Impl
03384
03385 } // namespace Matchers
03386
03387 using namespace Matchers;
03388 using Matchers::Impl::MatcherBase;
03389
03390 } // namespace Catch
03391
03392 // end catch_matchers.h
03393 // start catch_matchers_exception.hpp
03394
03395 namespace Catch {
03396     namespace Matchers {
03397         namespace Exception {
03398
03399             class ExceptionMessageMatcher : public MatcherBase<std::exception> {
03400             public:
03401                 std::string m_message;
03402
03403                 ExceptionMessageMatcher( std::string const& message ) :
03404                     m_message( message )
03405                 {}
03406
03407                 bool match( std::exception const& ex ) const override;
03408
03409                 std::string describe() const override;
03410             };
03411
03412         } // namespace Exception
03413
03414         Exception::ExceptionMessageMatcher Message( std::string const& message );
03415
03416     } // namespace Matchers
03417 } // namespace Catch
03418
03419 // end catch_matchers_exception.hpp

```



```

03507     }
03508
03509     std::string describe() const override {
03510         return m_description;
03511     }
03512 };
03513
03514 } // namespace Generic
03515
03516 // The following functions create the actual matcher objects.
03517 // The user has to explicitly specify type to the function, because
03518 // inferring std::function<bool(T const*)> is hard (but possible) and
03519 // requires a lot of TMP.
03520 template<typename T>
03521 Generic::PredicateMatcher<T> Predicate(std::function<bool(T const*)> const& predicate, std::string
const& description = "") {
03522     return Generic::PredicateMatcher<T>(predicate, description);
03523 }
03524
03525 } // namespace Matchers
03526 } // namespace Catch
03527
03528 // end catch_matchers_generic.hpp
03529 // start catch_matchers_string.h
03530
03531 #include <string>
03532
03533 namespace Catch {
03534     namespace Matchers {
03535         namespace StdString {
03536             struct CasedString
03537             {
03538                 CasedString( std::string const& str, CaseSensitive::Choice caseSensitivity );
03539                 std::string adjustString( std::string const& str ) const;
03540                 std::string caseSensitivitySuffix() const;
03541
03542                 CaseSensitive::Choice m_caseSensitivity;
03543                 std::string m_str;
03544             };
03545
03546             struct StringMatcherBase : MatcherBase<std::string> {
03547                 StringMatcherBase( std::string const& operation, CasedString const& comparator );
03548                 std::string describe() const override;
03549
03550                 CasedString m_comparator;
03551                 std::string m_operation;
03552             };
03553
03554             struct EqualsMatcher : StringMatcherBase {
03555                 EqualsMatcher( CasedString const& comparator );
03556                 bool match( std::string const& source ) const override;
03557             };
03558             struct ContainsMatcher : StringMatcherBase {
03559                 ContainsMatcher( CasedString const& comparator );
03560                 bool match( std::string const& source ) const override;
03561             };
03562             struct StartsWithMatcher : StringMatcherBase {
03563                 StartsWithMatcher( CasedString const& comparator );
03564                 bool match( std::string const& source ) const override;
03565             };
03566             struct EndsWithMatcher : StringMatcherBase {
03567                 EndsWithMatcher( CasedString const& comparator );
03568                 bool match( std::string const& source ) const override;
03569             };
03570
03571             struct RegexMatcher : MatcherBase<std::string> {
03572                 RegexMatcher( std::string regex, CaseSensitive::Choice caseSensitivity );
03573                 bool match( std::string const& matchee ) const override;
03574                 std::string describe() const override;
03575
03576             private:
03577                 std::string m_regex;
03578                 CaseSensitive::Choice m_caseSensitivity;
03579             };
03580         } // namespace StdString
03581
03582 // The following functions create the actual matcher objects.
03583 // This allows the types to be inferred
03584
03585 StdString::EqualsMatcher Equals( std::string const& str, CaseSensitive::Choice caseSensitivity =
CaseSensitive::Yes );
03586 StdString::ContainsMatcher Contains( std::string const& str, CaseSensitive::Choice caseSensitivity
= CaseSensitive::Yes );
03587 StdString::EndsWithMatcher EndsWith( std::string const& str, CaseSensitive::Choice caseSensitivity

```

```

    = CaseSensitive::Yes );
03591     StdString::StartsWithMatcher StartsWith( std::string const& str, CaseSensitive::Choice
caseSensitivity = CaseSensitive::Yes );
03592     StdString::RegexMatcher Matches( std::string const& regex, CaseSensitive::Choice caseSensitivity =
CaseSensitive::Yes );
03593
03594 } // namespace Matchers
03595 } // namespace Catch
03596
03597 // end catch_matchers_string.h
03598 // start catch_matchers_vector.h
03599
03600 #include <algorithm>
03601
03602 namespace Catch {
03603 namespace Matchers {
03604
03605     namespace Vector {
03606         template<typename T, typename Alloc>
03607         struct ContainsElementMatcher : MatcherBase<std::vector<T, Alloc> {
03608
03609             ContainsElementMatcher(T const& comparator) : m_comparator( comparator ) {}
03610
03611             bool match(std::vector<T, Alloc> const& v) const override {
03612                 for (auto const& el : v) {
03613                     if (el == m_comparator) {
03614                         return true;
03615                     }
03616                 }
03617                 return false;
03618             }
03619
03620             std::string describe() const override {
03621                 return "Contains: " + ::Catch::Detail::stringify( m_comparator );
03622             }
03623
03624             T const& m_comparator;
03625         };
03626
03627         template<typename T, typename AllocComp, typename AllocMatch>
03628         struct ContainsMatcher : MatcherBase<std::vector<T, AllocMatch> {
03629
03630             ContainsMatcher(std::vector<T, AllocComp> const& comparator) : m_comparator( comparator )
03631             {}
03632
03633             bool match(std::vector<T, AllocMatch> const& v) const override {
03634                 // !TBD: see note in EqualsMatcher
03635                 if (m_comparator.size() > v.size())
03636                     return false;
03637                 for (auto const& comparator : m_comparator) {
03638                     auto present = false;
03639                     for (const auto& el : v) {
03640                         if (el == comparator) {
03641                             present = true;
03642                             break;
03643                         }
03644                     }
03645                     if (!present) {
03646                         return false;
03647                     }
03648                 }
03649                 return true;
03650             }
03651             std::string describe() const override {
03652                 return "Contains: " + ::Catch::Detail::stringify( m_comparator );
03653             }
03654
03655             std::vector<T, AllocComp> const& m_comparator;
03656         };
03657
03658         template<typename T, typename AllocComp, typename AllocMatch>
03659         struct EqualsMatcher : MatcherBase<std::vector<T, AllocMatch> {
03660
03661             EqualsMatcher(std::vector<T, AllocComp> const& comparator) : m_comparator( comparator ) {}
03662
03663             bool match(std::vector<T, AllocMatch> const& v) const override {
03664                 // !TBD: This currently works if all elements can be compared using !=
03665                 // - a more general approach would be via a compare template that defaults
03666                 // to using !=. but could be specialised for, e.g. std::vector<T, Alloc> etc
03667                 // - then just call that directly
03668                 if (m_comparator.size() != v.size())
03669                     return false;
03670                 for (std::size_t i = 0; i < v.size(); ++i)
03671                     if (m_comparator[i] != v[i])
03672                         return false;
03673                 return true;
03674             }
03675         };
03676     }
03677 }

```

```

03674         std::string describe() const override {
03675             return "Equals: " + ::Catch::Detail::stringify( m_comparator );
03676         }
03677         std::vector<T, AllocComp> const& m_comparator;
03678     };
03679
03680     template<typename T, typename AllocComp, typename AllocMatch>
03681     struct ApproxMatcher : MatcherBase<std::vector<T, AllocMatch> > {
03682
03683         ApproxMatcher(std::vector<T, AllocComp> const& comparator) : m_comparator( comparator ) {}
03684
03685         bool match(std::vector<T, AllocMatch> const& v) const override {
03686             if (m_comparator.size() != v.size())
03687                 return false;
03688             for (std::size_t i = 0; i < v.size(); ++i)
03689                 if (m_comparator[i] != approx(v[i]))
03690                     return false;
03691             return true;
03692         }
03693         std::string describe() const override {
03694             return "is approx: " + ::Catch::Detail::stringify( m_comparator );
03695         }
03696     }
03697
03698     template <typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
03699     ApproxMatcher& epsilon( T const& newEpsilon ) {
03700         approx.epsilon(newEpsilon);
03701         return *this;
03702     }
03703
03704     template <typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
03705     ApproxMatcher& margin( T const& newMargin ) {
03706         approx.margin(newMargin);
03707         return *this;
03708     }
03709
03710     template <typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
03711     ApproxMatcher& scale( T const& newScale ) {
03712         approx.scale(newScale);
03713         return *this;
03714     }
03715
03716     std::vector<T, AllocComp> const& m_comparator;
03717     mutable Catch::Detail::Approx approx = Catch::Detail::Approx::custom();
03718 };
03719
03720     template<typename T, typename AllocComp, typename AllocMatch>
03721     struct UnorderedEqualsMatcher : MatcherBase<std::vector<T, AllocMatch> > {
03722         UnorderedEqualsMatcher(std::vector<T, AllocComp> const& target) : m_target(target) {}
03723         bool match(std::vector<T, AllocMatch> const& vec) const override {
03724             if (m_target.size() != vec.size()) {
03725                 return false;
03726             }
03727             return std::is_permutation(m_target.begin(), m_target.end(), vec.begin());
03728         }
03729         std::string describe() const override {
03730             return "UnorderedEquals: " + ::Catch::Detail::stringify(m_target);
03731         }
03732     private:
03733         std::vector<T, AllocComp> const& m_target;
03734     };
03735
03736     } // namespace Vector
03737
03738     // The following functions create the actual matcher objects.
03739     // This allows the types to be inferred
03740
03741     template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
03742     Vector::ContainsMatcher<T, AllocComp, AllocMatch> Contains( std::vector<T, AllocComp> const&
03743 comparator ) {
03744         return Vector::ContainsMatcher<T, AllocComp, AllocMatch>( comparator );
03745     }
03746
03747     template<typename T, typename Alloc = std::allocator<T> >
03748     Vector::ContainsElementMatcher<T, Alloc> VectorContains( T const& comparator ) {
03749         return Vector::ContainsElementMatcher<T, Alloc>( comparator );
03750     }
03751
03752     template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
03753     Vector::EqualsMatcher<T, AllocComp, AllocMatch> Equals( std::vector<T, AllocComp> const&
03754 comparator ) {
03755         return Vector::EqualsMatcher<T, AllocComp, AllocMatch>( comparator );
03756     }
03757
03758     template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
03759     Vector::ApproxMatcher<T, AllocComp, AllocMatch> Approx( std::vector<T, AllocComp> const&
03760 comparator ) {

```

```

03755         return Vector::ApproxMatcher<T, AllocComp, AllocMatch>( comparator );
03756     }
03757
03758     template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
03759     Vector::UnorderedEqualsMatcher<T, AllocComp, AllocMatch> UnorderedEquals( std::vector<T, AllocComp>
const& target ) {
03760         return Vector::UnorderedEqualsMatcher<T, AllocComp, AllocMatch>( target );
03761     }
03762
03763 } // namespace Matchers
03764 } // namespace Catch
03765
03766 // end catch_matchers_vector.h
03767 namespace Catch {
03768
03769     template<typename ArgT, typename MatcherT>
03770     class MatchExpr : public ITransientExpression {
03771     public:
03772         ArgT const& m_arg;
03773         MatcherT m_matcher;
03774         StringRef m_matcherString;
03775     public:
03776         MatchExpr( ArgT const& arg, MatcherT const& matcher, StringRef const& matcherString )
03777             : ITransientExpression{ true, matcher.match( arg ) },
03778               m_arg( arg ),
03779               m_matcher( matcher ),
03780               m_matcherString( matcherString )
03781         {}
03782
03783         void streamReconstructedExpression( std::ostream &os ) const override {
03784             auto matcherAsString = m_matcher.toString();
03785             os << Catch::Detail::stringify( m_arg ) << ' ';
03786             if( matcherAsString == Detail::unprintableString )
03787                 os << m_matcherString;
03788             else
03789                 os << matcherAsString;
03790         }
03791     };
03792
03793     using StringMatcher = Matchers::Impl::MatcherBase<std::string>;
03794
03795     void handleExceptionMatchExpr( AssertionHandler& handler, StringMatcher const& matcher, StringRef
const& matcherString );
03796
03797     template<typename ArgT, typename MatcherT>
03798     auto makeMatchExpr( ArgT const& arg, MatcherT const& matcher, StringRef const& matcherString ) ->
MatchExpr<ArgT, MatcherT> {
03799         return MatchExpr<ArgT, MatcherT>( arg, matcher, matcherString );
03800     }
03801 } // namespace Catch
03802
03803 #define INTERNAL_CHECK_THAT( macroName, matcher, resultDisposition, arg ) \
03804     do { \
03805         Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, \
CATCH_INTERNAL_STRINGIFY(arg) ", " CATCH_INTERNAL_STRINGIFY(matcher), resultDisposition ); \
03806         INTERNAL_CATCH_TRY { \
03807             catchAssertionHandler.handleExpr( Catch::makeMatchExpr( arg, matcher, #matcher##_catch_sr \
) ); \
03808         } INTERNAL_CATCH_CATCH( catchAssertionHandler ) \
03809         INTERNAL_CATCH_REACT( catchAssertionHandler ) \
03810     } while( false )
03811
03812 #define INTERNAL_CATCH_THROWS_MATCHES( macroName, exceptionType, resultDisposition, matcher, ... ) \
03813     do { \
03814         Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, \
CATCH_INTERNAL_STRINGIFY(__VA_ARGS__) ", " CATCH_INTERNAL_STRINGIFY(exceptionType) ", " \
CATCH_INTERNAL_STRINGIFY(matcher), resultDisposition ); \
03815         if( catchAssertionHandler.allowThrows() ) \
03816             try { \
03817                 static_cast<void>( __VA_ARGS__ ); \
03818                 catchAssertionHandler.handleUnexpectedExceptionNotThrown(); \
03819             } \
03820             catch( exceptionType const& ex ) { \
03821                 catchAssertionHandler.handleExpr( Catch::makeMatchExpr( ex, matcher, \
#matcher##_catch_sr ) ); \
03822             } \
03823             catch( ... ) { \
03824                 catchAssertionHandler.handleUnexpectedInflightException(); \
03825             } \
03826         else \
03827             catchAssertionHandler.handleThrowingCallSkipped(); \
03828         INTERNAL_CATCH_REACT( catchAssertionHandler ) \
03829     } while( false )
03830
03831 // end catch_capture_matchers.h
03832 #endif
03833 // start catch_generators.hpp

```

```

03836
03837 // start catch_interfaces_generatortracker.h
03838
03839
03840 #include <memory>
03841
03842 namespace Catch {
03843
03844     namespace Generators {
03845         class GeneratorUntypedBase {
03846         public:
03847             GeneratorUntypedBase() = default;
03848             virtual ~GeneratorUntypedBase();
03849             // Attempts to move the generator to the next element
03850             //
03851             // Returns true iff the move succeeded (and a valid element
03852             // can be retrieved).
03853             virtual bool next() = 0;
03854         };
03855         using GeneratorBasePtr = std::unique_ptr<GeneratorUntypedBase>;
03856
03857     } // namespace Generators
03858
03859     struct IGeneratorTracker {
03860     public:
03861         virtual ~IGeneratorTracker();
03862         virtual auto hasGenerator() const -> bool = 0;
03863         virtual auto getGenerator() const -> Generators::GeneratorBasePtr const& = 0;
03864         virtual void setGenerator( Generators::GeneratorBasePtr&& generator ) = 0;
03865     };
03866 } // namespace Catch
03867
03868 // end catch_interfaces_generatortracker.h
03869 // start catch_enforce.h
03870
03871 #include <exception>
03872
03873 namespace Catch {
03874 #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
03875     template <typename Ex>
03876     [[noreturn]]
03877     void throw_exception(Ex const& e) {
03878         throw e;
03879     }
03880 #else // ^^ Exceptions are enabled // Exceptions are disabled vv
03881     [[noreturn]]
03882     void throw_exception(std::exception const& e);
03883 #endif
03884
03885     [[noreturn]]
03886     void throw_logic_error(std::string const& msg);
03887     [[noreturn]]
03888     void throw_domain_error(std::string const& msg);
03889     [[noreturn]]
03890     void throw_runtime_error(std::string const& msg);
03891 } // namespace Catch;
03892
03893 #define CATCH_MAKE_MSG(...) \
03894     (Catch::ReusableStringStream() << __VA_ARGS__).str()
03895
03896 #define CATCH_INTERNAL_ERROR(...) \
03897     Catch::throw_logic_error(CATCH_MAKE_MSG( CATCH_INTERNAL_LINEINFO << "Internal Catch2 error: " << \
03898     __VA_ARGS__))
03899
03900 #define CATCH_ERROR(...) \
03901     Catch::throw_domain_error(CATCH_MAKE_MSG( __VA_ARGS__ ))
03902
03903 #define CATCH_RUNTIME_ERROR(...) \
03904     Catch::throw_runtime_error(CATCH_MAKE_MSG( __VA_ARGS__ ))
03905
03906 #define CATCH_ENFORCE( condition, ... ) \
03907     do{ if( !(condition) ) CATCH_ERROR( __VA_ARGS__ ); } while(false)
03908
03909 // end catch_enforce.h
03910 #include <memory>
03911 #include <vector>
03912 #include <cassert>
03913
03914 #include <utility>
03915 #include <exception>
03916
03917 namespace Catch {
03918
03919     class GeneratorException : public std::exception {
03920     public:
03921         const char* const m_msg = "";
03922     };

```

```

03922 public:
03923     GeneratorException(const char* msg):
03924         m_msg(msg)
03925     {}
03926
03927     const char* what() const noexcept override final;
03928 };
03929
03930 namespace Generators {
03931
03932     // !TBD move this into its own location?
03933     namespace pf{
03934         template<typename T, typename... Args>
03935         std::unique_ptr<T> make_unique( Args&&... args ) {
03936             return std::unique_ptr<T>(new T(std::forward<Args>(args)...));
03937         }
03938     }
03939
03940     template<typename T>
03941     struct IGenerator : GeneratorUntypedBase {
03942         virtual ~IGenerator() = default;
03943
03944         // Returns the current element of the generator
03945         //
03946         // \Precondition The generator is either freshly constructed,
03947         // or the last call to `next()` returned true
03948         virtual T const& get() const = 0;
03949         using type = T;
03950     };
03951
03952     template<typename T>
03953     class SingleValueGenerator final : public IGenerator<T> {
03954         T m_value;
03955     public:
03956         SingleValueGenerator(T&& value) : m_value(std::move(value)) {}
03957
03958         T const& get() const override {
03959             return m_value;
03960         }
03961         bool next() override {
03962             return false;
03963         }
03964     };
03965
03966     template<typename T>
03967     class FixedValuesGenerator final : public IGenerator<T> {
03968         static_assert(!std::is_same<T, bool>::value,
03969             "FixedValuesGenerator does not support bools because of std::vector<bool>"
03970             "specialization, use SingleValue Generator instead.");
03971         std::vector<T> m_values;
03972         size_t m_idx = 0;
03973     public:
03974         FixedValuesGenerator( std::initializer_list<T> values ) : m_values( values ) {}
03975
03976         T const& get() const override {
03977             return m_values[m_idx];
03978         }
03979         bool next() override {
03980             ++m_idx;
03981             return m_idx < m_values.size();
03982         }
03983     };
03984
03985     template <typename T>
03986     class GeneratorWrapper final {
03987         std::unique_ptr<IGenerator<T> m_generator;
03988     public:
03989         GeneratorWrapper(std::unique_ptr<IGenerator<T> generator):
03990             m_generator(std::move(generator))
03991         {}
03992         T const& get() const {
03993             return m_generator->get();
03994         }
03995         bool next() {
03996             return m_generator->next();
03997         }
03998     };
03999
04000     template <typename T>
04001     GeneratorWrapper<T> value(T&& value) {
04002         return GeneratorWrapper<T>(pf::make_unique<SingleValueGenerator<T>>(std::forward<T>(value)));
04003     }
04004     template <typename T>
04005     GeneratorWrapper<T> values(std::initializer_list<T> values) {
04006         return GeneratorWrapper<T>(pf::make_unique<FixedValuesGenerator<T>>(values));
04007     }
04008

```



```

04009     template<typename T>
04010     class Generators : public IGenerator<T> {
04011     public:
04012         std::vector<GeneratorWrapper<T>> m_generators;
04013         size_t m_current = 0;
04014
04015         void populate(GeneratorWrapper<T>&& generator) {
04016             m_generators.emplace_back(std::move(generator));
04017         }
04018         void populate(T&& val) {
04019             m_generators.emplace_back(value(std::forward<T>(val)));
04020         }
04021         template<typename U>
04022         void populate(U&& val) {
04023             populate(T(std::forward<U>(val)));
04024         }
04025         template<typename U, typename... Gs>
04026         void populate(U&& valueOrGenerator, Gs &&... moreGenerators) {
04027             populate(std::forward<U>(valueOrGenerator));
04028             populate(std::forward<Gs>(moreGenerators)...);
04029         }
04030     public:
04031         template<typename... Gs>
04032         Generators(Gs &&... moreGenerators) {
04033             m_generators.reserve(sizeof...(Gs));
04034             populate(std::forward<Gs>(moreGenerators)...);
04035         }
04036
04037         T const& get() const override {
04038             return m_generators[m_current].get();
04039         }
04040
04041         bool next() override {
04042             if (m_current >= m_generators.size()) {
04043                 return false;
04044             }
04045             const bool current_status = m_generators[m_current].next();
04046             if (!current_status) {
04047                 ++m_current;
04048             }
04049             return m_current < m_generators.size();
04050         }
04051     };
04052
04053     template<typename... Ts>
04054     GeneratorWrapper<std::tuple<Ts...>> table( std::initializer_list<std::tuple<typename
std::decay<Ts>::type...> tuples ) {
04055         return values<std::tuple<Ts...>>( tuples );
04056     }
04057
04058     // Tag type to signal that a generator sequence should convert arguments to a specific type
04059     template<typename T>
04060     struct as {};
04061
04062     template<typename T, typename... Gs>
04063     auto makeGenerators( GeneratorWrapper<T>&& generator, Gs &&... moreGenerators ) -> Generators<T> {
04064         return Generators<T>(std::move(generator), std::forward<Gs>(moreGenerators)...);
04065     }
04066     template<typename T>
04067     auto makeGenerators( GeneratorWrapper<T>&& generator ) -> Generators<T> {
04068         return Generators<T>(std::move(generator));
04069     }
04070     template<typename T, typename... Gs>
04071     auto makeGenerators( T&& val, Gs &&... moreGenerators ) -> Generators<T> {
04072         return makeGenerators( value( std::forward<T>( val ) ), std::forward<Gs>( moreGenerators )...
);
04073     }
04074     template<typename T, typename U, typename... Gs>
04075     auto makeGenerators( as<T>, U&& val, Gs &&... moreGenerators ) -> Generators<T> {
04076         return makeGenerators( value( T( std::forward<U>( val ) ) ), std::forward<Gs>( moreGenerators
)... );
04077     }
04078
04079     auto acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo ) ->
IGeneratorTracker&;
04080
04081     template<typename L>
04082     // Note: The type after -> is weird, because VS2015 cannot parse
04083     // the expression used in the typedef inside, when it is in
04084     // return type. Yeah.
04085     auto generate( StringRef generatorName, SourceLineInfo const& lineInfo, L const&
generatorExpression ) -> decltype( std::declval<decltype(generatorExpression())>().get() ) {
04086         using UnderlyingType = typename decltype(generatorExpression())::type;
04087
04088         IGeneratorTracker& tracker = acquireGeneratorTracker( generatorName, lineInfo );
04089         if (!tracker.hasGenerator()) {
04090             tracker.setGenerator(pf::make_unique<Generators<UnderlyingType>>(generatorExpression()));

```

```

04091     }
04092
04093     auto const& generator = static_cast<IGenerator<UnderlyingType> const&>(
*tracker.getGenerator() );
04094     return generator.get();
04095 }
04096
04097 } // namespace Generators
04098 } // namespace Catch
04099
04100 #define GENERATE( ... ) \
04101     Catch::Generators::generate( INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_UNIQUE_NAME(generator)), \
04102                                 CATCH_INTERNAL_LINEINFO, \
04103                                 [ ]{ using namespace Catch::Generators; return makeGenerators(
__VA_ARGS__ ); } ) //NOLINT(google-build-using-namespace)
04104 #define GENERATE_COPY( ... ) \
04105     Catch::Generators::generate( INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_UNIQUE_NAME(generator)), \
04106                                 CATCH_INTERNAL_LINEINFO, \
04107                                 [=]{ using namespace Catch::Generators; return makeGenerators(
__VA_ARGS__ ); } ) //NOLINT(google-build-using-namespace)
04108 #define GENERATE_REF( ... ) \
04109     Catch::Generators::generate( INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_UNIQUE_NAME(generator)), \
04110                                 CATCH_INTERNAL_LINEINFO, \
04111                                 [&]{ using namespace Catch::Generators; return makeGenerators(
__VA_ARGS__ ); } ) //NOLINT(google-build-using-namespace)
04112
04113 // end catch_generators.hpp
04114 // start catch_generators_generic.hpp
04115
04116 namespace Catch {
04117 namespace Generators {
04118
04119     template <typename T>
04120     class TakeGenerator : public IGenerator<T> {
04121     public:
04122         GeneratorWrapper<T> m_generator;
04123         size_t m_returned = 0;
04124         size_t m_target;
04125
04126         TakeGenerator(size_t target, GeneratorWrapper<T>&& generator):
04127             m_generator(std::move(generator)),
04128             m_target(target)
04129         {
04130             assert(target != 0 && "Empty generators are not allowed");
04131
04132             T const& get() const override {
04133                 return m_generator.get();
04134             }
04135
04136             bool next() override {
04137                 ++m_returned;
04138                 if (m_returned >= m_target) {
04139                     return false;
04140                 }
04141
04142                 const auto success = m_generator.next();
04143                 // If the underlying generator does not contain enough values
04144                 // then we cut short as well
04145                 if (!success) {
04146                     m_returned = m_target;
04147                 }
04148                 return success;
04149             }
04150
04151             template <typename T>
04152             GeneratorWrapper<T> take(size_t target, GeneratorWrapper<T>&& generator) {
04153                 return GeneratorWrapper<T>(pf::make_unique<TakeGenerator<T>>(target, std::move(generator)));
04154             }
04155
04156             template <typename T, typename Predicate>
04157             class FilterGenerator : public IGenerator<T> {
04158             public:
04159                 GeneratorWrapper<T> m_generator;
04160                 Predicate m_predicate;
04161
04162                 FilterGenerator(P&& pred, GeneratorWrapper<T>&& generator):
04163                     m_generator(std::move(generator)),
04164                     m_predicate(std::forward<P>(pred))
04165                 {
04166                     if (!m_predicate(m_generator.get())) {
04167                         // It might happen that there are no values that pass the
04168                         // filter. In that case we throw an exception.
04169                         auto has_initial_value = nextImpl();
04170                         if (!has_initial_value) {
04171                             Catch::throw_exception(GeneratorException("No valid value found in filtered
generator"));
04172                         }
04173                     }
04174                 }
04175
04176                 T const& get() const override {
04177                     return m_generator.get();
04178                 }
04179
04180                 bool next() override {
04181                     return m_predicate(m_generator.get()) && m_generator.next();
04182                 }
04183             };
04184
04185             template <typename T, typename Predicate>
04186             FilterGenerator<T, Predicate> filter(P&& pred, GeneratorWrapper<T>&& generator) {
04187                 return FilterGenerator<T, Predicate>(pred, std::move(generator));
04188             }
04189         };
04190     };
04191 }
04192 }

```

```

04173     }
04174
04175     T const& get() const override {
04176         return m_generator.get();
04177     }
04178
04179     bool next() override {
04180         return nextImpl();
04181     }
04182
04183 private:
04184     bool nextImpl() {
04185         bool success = m_generator.next();
04186         if (!success) {
04187             return false;
04188         }
04189         while (!m_predicate(m_generator.get()) && (success = m_generator.next()) == true);
04190         return success;
04191     }
04192 };
04193
04194 template <typename T, typename Predicate>
04195 GeneratorWrapper<T> filter(Predicate&& pred, GeneratorWrapper<T>&& generator) {
04196     return GeneratorWrapper<T>(std::unique_ptr<IGenerator<T>>(pf::make_unique<FilterGenerator<T, Predicate>>(std::forward<Predicate>(pred), std::move(generator)))));
04197 }
04198
04199 template <typename T>
04200 class RepeatGenerator : public IGenerator<T> {
04201     static_assert(!std::is_same<T, bool>::value,
04202         "RepeatGenerator currently does not support bools"
04203         "because of std::vector<bool> specialization");
04204     GeneratorWrapper<T> m_generator;
04205     mutable std::vector<T> m_returned;
04206     size_t m_target_repeats;
04207     size_t m_current_repeat = 0;
04208     size_t m_repeat_index = 0;
04209 public:
04210     RepeatGenerator(size_t repeats, GeneratorWrapper<T>&& generator):
04211         m_generator(std::move(generator)),
04212         m_target_repeats(repeats)
04213     {
04214         assert(m_target_repeats > 0 && "Repeat generator must repeat at least once");
04215     }
04216
04217     T const& get() const override {
04218         if (m_current_repeat == 0) {
04219             m_returned.push_back(m_generator.get());
04220             return m_returned.back();
04221         }
04222         return m_returned[m_repeat_index];
04223     }
04224
04225     bool next() override {
04226         // There are 2 basic cases:
04227         // 1) We are still reading the generator
04228         // 2) We are reading our own cache
04229
04230         // In the first case, we need to poke the underlying generator.
04231         // If it happily moves, we are left in that state, otherwise it is time to start reading
04232         from our cache
04233         if (m_current_repeat == 0) {
04234             const auto success = m_generator.next();
04235             if (!success) {
04236                 ++m_current_repeat;
04237             }
04238             return m_current_repeat < m_target_repeats;
04239         }
04240
04241         // In the second case, we need to move indices forward and check that we haven't run up
04242         against the end
04243         ++m_repeat_index;
04244         if (m_repeat_index == m_returned.size()) {
04245             m_repeat_index = 0;
04246             ++m_current_repeat;
04247         }
04248         return m_current_repeat < m_target_repeats;
04249     }
04250 };
04251
04252 template <typename T>
04253 GeneratorWrapper<T> repeat(size_t repeats, GeneratorWrapper<T>&& generator) {
04254     return GeneratorWrapper<T>(pf::make_unique<RepeatGenerator<T>>(repeats, std::move(generator)));
04255 }

```

```

04255     template <typename T, typename U, typename Func>
04256     class MapGenerator : public IGenerator<T> {
04257     // TBD: provide static assert for mapping function, for friendly error message
04258         GeneratorWrapper<U> m_generator;
04259         Func m_function;
04260         // To avoid returning dangling reference, we have to save the values
04261         T m_cache;
04262     public:
04263         template <typename F2 = Func>
04264         MapGenerator(F2&& function, GeneratorWrapper<U>&& generator) :
04265             m_generator(std::move(generator)),
04266             m_function(std::forward<F2>(function)),
04267             m_cache(m_function(m_generator.get()))
04268         {}
04269
04270         T const& get() const override {
04271             return m_cache;
04272         }
04273         bool next() override {
04274             const auto success = m_generator.next();
04275             if (success) {
04276                 m_cache = m_function(m_generator.get());
04277             }
04278             return success;
04279         }
04280     };
04281
04282     template <typename Func, typename U, typename T = FunctionReturnType<Func, U>
04283     GeneratorWrapper<T> map(Func&& function, GeneratorWrapper<U>&& generator) {
04284         return GeneratorWrapper<T>(
04285             pf::make_unique<MapGenerator<T, U, Func>>(std::forward<Func>(function),
04286             std::move(generator))
04287         );
04288
04289     template <typename T, typename U, typename Func>
04290     GeneratorWrapper<T> map(Func&& function, GeneratorWrapper<U>&& generator) {
04291         return GeneratorWrapper<T>(
04292             pf::make_unique<MapGenerator<T, U, Func>>(std::forward<Func>(function),
04293             std::move(generator))
04294         );
04295
04296     template <typename T>
04297     class ChunkGenerator final : public IGenerator<std::vector<T>> {
04298     std::vector<T> m_chunk;
04299     size_t m_chunk_size;
04300     GeneratorWrapper<T> m_generator;
04301     bool m_used_up = false;
04302     public:
04303     ChunkGenerator(size_t size, GeneratorWrapper<T> generator) :
04304         m_chunk_size(size), m_generator(std::move(generator))
04305     {
04306         m_chunk.reserve(m_chunk_size);
04307         if (m_chunk_size != 0) {
04308             m_chunk.push_back(m_generator.get());
04309             for (size_t i = 1; i < m_chunk_size; ++i) {
04310                 if (!m_generator.next()) {
04311                     Catch::throw_exception(GeneratorException("Not enough values to initialize the
first chunk"));
04312                 }
04313                 m_chunk.push_back(m_generator.get());
04314             }
04315         }
04316
04317         std::vector<T> const& get() const override {
04318             return m_chunk;
04319         }
04320         bool next() override {
04321             m_chunk.clear();
04322             for (size_t idx = 0; idx < m_chunk_size; ++idx) {
04323                 if (!m_generator.next()) {
04324                     return false;
04325                 }
04326                 m_chunk.push_back(m_generator.get());
04327             }
04328             return true;
04329         }
04330     };
04331
04332     template <typename T>
04333     GeneratorWrapper<std::vector<T>> chunk(size_t size, GeneratorWrapper<T>&& generator) {
04334         return GeneratorWrapper<std::vector<T>>(
04335             pf::make_unique<ChunkGenerator<T>>(size, std::move(generator))
04336         );
04337     }
04338

```

```

04339 } // namespace Generators
04340 } // namespace Catch
04341
04342 // end catch_generators_generic.hpp
04343 // start catch_generators_specific.hpp
04344
04345 // start catch_context.h
04346
04347 #include <memory>
04348
04349 namespace Catch {
04350
04351     struct IResultCapture;
04352     struct IRunner;
04353     struct IConfig;
04354     struct IMutableContext;
04355
04356     using IConfigPtr = std::shared_ptr<IConfig const>;
04357
04358     struct IContext
04359     {
04360         virtual ~IContext();
04361
04362         virtual IResultCapture* getResultCapture() = 0;
04363         virtual IRunner* getRunner() = 0;
04364         virtual IConfigPtr const& getConfig() const = 0;
04365     };
04366
04367     struct IMutableContext : IContext
04368     {
04369         virtual ~IMutableContext();
04370         virtual void setResultCapture( IResultCapture* resultCapture ) = 0;
04371         virtual void setRunner( IRunner* runner ) = 0;
04372         virtual void setConfig( IConfigPtr const& config ) = 0;
04373
04374     private:
04375         static IMutableContext* currentContext;
04376         friend IMutableContext& getCurrentMutableContext();
04377         friend void cleanUpContext();
04378         static void createContext();
04379     };
04380
04381     inline IMutableContext& getCurrentMutableContext()
04382     {
04383         if( !IMutableContext::currentContext )
04384             IMutableContext::createContext();
04385         // NOLINTNEXTLINE(clang-analyzer-core.uninitialized.UndefReturn)
04386         return *IMutableContext::currentContext;
04387     }
04388
04389     inline IContext& getCurrentContext()
04390     {
04391         return getCurrentMutableContext();
04392     }
04393
04394     void cleanUpContext();
04395
04396     class SimplePcg32;
04397     SimplePcg32& rng();
04398 }
04399
04400 // end catch_context.h
04401 // start catch_interfaces_config.h
04402
04403 // start catch_option.hpp
04404
04405 namespace Catch {
04406
04407     // An optional type
04408     template<typename T>
04409     class Option {
04410     public:
04411         Option() : nullableValue( nullptr ) {}
04412         Option( T const& _value )
04413             : nullableValue( new( storage ) T( _value ) )
04414             {}
04415         Option( Option const& _other )
04416             : nullableValue( _other ? new( storage ) T( *_other ) : nullptr )
04417             {}
04418
04419         ~Option() {
04420             reset();
04421         }
04422
04423         Option& operator= ( Option const& _other ) {
04424             if( &_other != this ) {
04425                 reset();

```

```

04426         if( _other )
04427             nullableValue = new( storage ) T( *_other );
04428     }
04429     return *this;
04430 }
04431 Option& operator = ( T const& _value ) {
04432     reset();
04433     nullableValue = new( storage ) T( _value );
04434     return *this;
04435 }
04436
04437 void reset() {
04438     if( nullableValue )
04439         nullableValue->~T();
04440     nullableValue = nullptr;
04441 }
04442
04443 T& operator*() { return *nullableValue; }
04444 T const& operator*() const { return *nullableValue; }
04445 T* operator->() { return nullableValue; }
04446 const T* operator->() const { return nullableValue; }
04447
04448 T valueOr( T const& defaultValue ) const {
04449     return nullableValue ? *nullableValue : defaultValue;
04450 }
04451
04452 bool some() const { return nullableValue != nullptr; }
04453 bool none() const { return nullableValue == nullptr; }
04454
04455 bool operator !() const { return nullableValue == nullptr; }
04456 explicit operator bool() const {
04457     return some();
04458 }
04459
04460 private:
04461     T *nullableValue;
04462     alignas(alignof(T)) char storage[sizeof(T)];
04463 };
04464
04465 } // end namespace Catch
04466
04467 // end catch_option.hpp
04468 #include <chrono>
04469 #include <iosfwd>
04470 #include <string>
04471 #include <vector>
04472 #include <memory>
04473
04474 namespace Catch {
04475
04476     enum class Verbosity {
04477         Quiet = 0,
04478         Normal,
04479         High
04480     };
04481
04482     struct WarnAbout { enum What {
04483         Nothing = 0x00,
04484         NoAssertions = 0x01,
04485         NoTests = 0x02
04486     }; };
04487
04488     struct ShowDurations { enum OrNot {
04489         DefaultForReporter,
04490         Always,
04491         Never
04492     }; };
04493
04494     struct RunTests { enum InWhatOrder {
04495         InDeclarationOrder,
04496         InLexicographicalOrder,
04497         InRandomOrder
04498     }; };
04499
04500     struct UseColour { enum YesOrNo {
04501         Auto,
04502         Yes,
04503         No
04504     }; };
04505
04506     struct WaitForKeypress { enum When {
04507         Never,
04508         BeforeStart = 1,
04509         BeforeExit = 2,
04510         BeforeStartAndExit = BeforeStart | BeforeExit
04511     }; };
04512
04513     class TestSpec;
04514
04515     struct IConfig : NonCopyable {

```

```

04513
04514     virtual ~IConfig();
04515
04516     virtual bool allowThrows() const = 0;
04517     virtual std::ostream& stream() const = 0;
04518     virtual std::string name() const = 0;
04519     virtual bool includeSuccessfulResults() const = 0;
04520     virtual bool shouldDebugBreak() const = 0;
04521     virtual bool warnAboutMissingAssertions() const = 0;
04522     virtual bool warnAboutNoTests() const = 0;
04523     virtual int abortAfter() const = 0;
04524     virtual bool showInvisibles() const = 0;
04525     virtual ShowDurations::OrNot showDurations() const = 0;
04526     virtual double minDuration() const = 0;
04527     virtual TestSpec const& testSpec() const = 0;
04528     virtual bool hasTestFilters() const = 0;
04529     virtual std::vector<std::string> const& getTestsOrTags() const = 0;
04530     virtual RunTests::InWhatOrder runOrder() const = 0;
04531     virtual unsigned int rngSeed() const = 0;
04532     virtual UseColour::YesOrNo useColour() const = 0;
04533     virtual std::vector<std::string> const& getSectionsToRun() const = 0;
04534     virtual Verbosity verbosity() const = 0;
04535
04536     virtual bool benchmarkNoAnalysis() const = 0;
04537     virtual int benchmarkSamples() const = 0;
04538     virtual double benchmarkConfidenceInterval() const = 0;
04539     virtual unsigned int benchmarkResamples() const = 0;
04540     virtual std::chrono::milliseconds benchmarkWarmupTime() const = 0;
04541 };
04542
04543 using IConfigPtr = std::shared_ptr<IConfig const>;
04544 }
04545
04546 // end catch_interfaces_config.h
04547 // start catch_random_number_generator.h
04548
04549 #include <stdint>
04550
04551 namespace Catch {
04552
04553     // This is a simple implementation of C++11 Uniform Random Number
04554     // Generator. It does not provide all operators, because Catch2
04555     // does not use it, but it should behave as expected inside stdlib's
04556     // distributions.
04557     // The implementation is based on the PCG family (http://pcg-random.org)
04558     class SimplePcg32 {
04559     public:
04560         using state_type = std::uint64_t;
04561         using result_type = std::uint32_t;
04562         static constexpr result_type (min)() {
04563             return 0;
04564         }
04565         static constexpr result_type (max)() {
04566             return static_cast<result_type>(-1);
04567         }
04568
04569         // Provide some default initial state for the default constructor
04570         SimplePcg32(): SimplePcg32(0xed743cc4U) {}
04571
04572         explicit SimplePcg32(result_type seed_);
04573
04574         void seed(result_type seed_);
04575         void discard(uint64_t skip);
04576
04577         result_type operator()();
04578
04579     private:
04580         friend bool operator==(SimplePcg32 const& lhs, SimplePcg32 const& rhs);
04581         friend bool operator!=(SimplePcg32 const& lhs, SimplePcg32 const& rhs);
04582
04583         // In theory we also need operator< and operator>
04584         // In practice we do not use them, so we will skip them for now
04585
04586         std::uint64_t m_state;
04587         // This part of the state determines which "stream" of the numbers
04588         // is chosen -- we take it as a constant for Catch2, so we only
04589         // need to deal with seeding the main state.
04590         // Picked by reading 8 bytes from `/dev/random` :-)
04591         static const std::uint64_t s_inc = (0x13ed0cc53f939476ULL << 1ULL) | 1ULL;
04592     };
04593
04594 } // end namespace Catch
04595
04596 // end catch_random_number_generator.h
04597 #include <random>
04598
04599 namespace Catch {

```

```

04600 namespace Generators {
04601
04602 template <typename Float>
04603 class RandomFloatingGenerator final : public IGenerator<Float> {
04604     Catch::SimplePcg32& m_rng;
04605     std::uniform_real_distribution<Float> m_dist;
04606     Float m_current_number;
04607 public:
04608
04609     RandomFloatingGenerator(Float a, Float b):
04610         m_rng(rng()),
04611         m_dist(a, b) {
04612         static_cast<void>(next());
04613     }
04614
04615     Float const& get() const override {
04616         return m_current_number;
04617     }
04618     bool next() override {
04619         m_current_number = m_dist(m_rng);
04620         return true;
04621     }
04622 };
04623
04624 template <typename Integer>
04625 class RandomIntegerGenerator final : public IGenerator<Integer> {
04626     Catch::SimplePcg32& m_rng;
04627     std::uniform_int_distribution<Integer> m_dist;
04628     Integer m_current_number;
04629 public:
04630
04631     RandomIntegerGenerator(Integer a, Integer b):
04632         m_rng(rng()),
04633         m_dist(a, b) {
04634         static_cast<void>(next());
04635     }
04636
04637     Integer const& get() const override {
04638         return m_current_number;
04639     }
04640     bool next() override {
04641         m_current_number = m_dist(m_rng);
04642         return true;
04643     }
04644 };
04645
04646 // TODO: Ideally this would be also constrained against the various char types,
04647 // but I don't expect users to run into that in practice.
04648 template <typename T>
04649 typename std::enable_if<std::is_integral<T>::value && !std::is_same<T, bool>::value,
04650 GeneratorWrapper<T>>::type
04651 random(T a, T b) {
04652     return GeneratorWrapper<T>(
04653         pf::make_unique<RandomIntegerGenerator<T>>(a, b)
04654     );
04655 }
04656
04657 template <typename T>
04658 typename std::enable_if<std::is_floating_point<T>::value,
04659 GeneratorWrapper<T>>::type
04660 random(T a, T b) {
04661     return GeneratorWrapper<T>(
04662         pf::make_unique<RandomFloatingGenerator<T>>(a, b)
04663     );
04664 }
04665
04666 template <typename T>
04667 class RangeGenerator final : public IGenerator<T> {
04668     T m_current;
04669     T m_end;
04670     T m_step;
04671     bool m_positive;
04672
04673 public:
04674     RangeGenerator(T const& start, T const& end, T const& step):
04675         m_current(start),
04676         m_end(end),
04677         m_step(step),
04678         m_positive(m_step > T(0))
04679     {
04680         assert(m_current != m_end && "Range start and end cannot be equal");
04681         assert(m_step != T(0) && "Step size cannot be zero");
04682         assert(((m_positive && m_current <= m_end) || (!m_positive && m_current >= m_end)) && "Step
moves away from end");
04683     }
04684
04685     RangeGenerator(T const& start, T const& end):

```



```

04686     RangeGenerator(start, end, (start < end) ? T(1) : T(-1))
04687     {}
04688
04689     T const& get() const override {
04690         return m_current;
04691     }
04692
04693     bool next() override {
04694         m_current += m_step;
04695         return (m_positive) ? (m_current < m_end) : (m_current > m_end);
04696     }
04697 };
04698
04699 template <typename T>
04700 GeneratorWrapper<T> range(T const& start, T const& end, T const& step) {
04701     static_assert(std::is_arithmetic<T>::value && !std::is_same<T, bool>::value, "Type must be
numeric");
04702     return GeneratorWrapper<T>(pf::make_unique<RangeGenerator<T>>(start, end, step));
04703 }
04704
04705 template <typename T>
04706 GeneratorWrapper<T> range(T const& start, T const& end) {
04707     static_assert(std::is_integral<T>::value && !std::is_same<T, bool>::value, "Type must be an
integer");
04708     return GeneratorWrapper<T>(pf::make_unique<RangeGenerator<T>>(start, end));
04709 }
04710
04711 template <typename T>
04712 class IteratorGenerator final : public IGenerator<T> {
04713     static_assert(!std::is_same<T, bool>::value,
04714         "IteratorGenerator currently does not support bools"
04715         "because of std::vector<bool> specialization");
04716
04717     std::vector<T> m_elems;
04718     size_t m_current = 0;
04719 public:
04720     template <typename InputIterator, typename InputSentinel>
04721     IteratorGenerator(InputIterator first, InputSentinel last):m_elems(first, last) {
04722         if (m_elems.empty()) {
04723             Catch::throw_exception(GeneratorException("IteratorGenerator received no valid values"));
04724         }
04725     }
04726
04727     T const& get() const override {
04728         return m_elems[m_current];
04729     }
04730
04731     bool next() override {
04732         ++m_current;
04733         return m_current != m_elems.size();
04734     }
04735 };
04736
04737 template <typename InputIterator,
04738     typename InputSentinel,
04739     typename ResultType = typename std::iterator_traits<InputIterator>::value_type>
04740 GeneratorWrapper<ResultType> from_range(InputIterator from, InputSentinel to) {
04741     return GeneratorWrapper<ResultType>(pf::make_unique<IteratorGenerator<ResultType>>(from, to));
04742 }
04743
04744 template <typename Container,
04745     typename ResultType = typename Container::value_type>
04746 GeneratorWrapper<ResultType> from_range(Container const& cnt) {
04747     return GeneratorWrapper<ResultType>(pf::make_unique<IteratorGenerator<ResultType>>(cnt.begin(),
cnt.end()));
04748 }
04749
04750 } // namespace Generators
04751 } // namespace Catch
04752
04753 // end catch_generators_specific.hpp
04754
04755 // These files are included here so the single_include script doesn't put them
04756 // in the conditionally compiled sections
04757 // start catch_test_case_info.h
04758
04759 #include <string>
04760 #include <vector>
04761 #include <memory>
04762
04763 #ifdef __clang__
04764 #pragma clang diagnostic push
04765 #pragma clang diagnostic ignored "-Wpadded"
04766 #endif
04767
04768 namespace Catch {
04769

```

```

04770     struct ITestInvoker;
04771
04772     struct TestCaseInfo {
04773         enum SpecialProperties{
04774             None = 0,
04775             IsHidden = 1 « 1,
04776             ShouldFail = 1 « 2,
04777             MayFail = 1 « 3,
04778             Throws = 1 « 4,
04779             NonPortable = 1 « 5,
04780             Benchmark = 1 « 6
04781         };
04782
04783         TestCaseInfo(    std::string const& _name,
04784                         std::string const& _className,
04785                         std::string const& _description,
04786                         std::vector<std::string> const& _tags,
04787                         SourceLineInfo const& _lineInfo );
04788
04789         friend void setTags( TestCaseInfo& testCaseInfo, std::vector<std::string> tags );
04790
04791         bool isHidden() const;
04792         bool throws() const;
04793         bool okToFail() const;
04794         bool expectedToFail() const;
04795
04796         std::string tagsAsString() const;
04797
04798         std::string name;
04799         std::string className;
04800         std::string description;
04801         std::vector<std::string> tags;
04802         std::vector<std::string> lcaseTags;
04803         SourceLineInfo lineInfo;
04804         SpecialProperties properties;
04805     };
04806
04807     class TestCase : public TestCaseInfo {
04808     public:
04809
04810         TestCase( ITestInvoker* testCase, TestCaseInfo&& info );
04811
04812         TestCase withName( std::string const& _newName ) const;
04813
04814         void invoke() const;
04815
04816         TestCaseInfo const& getTestCaseInfo() const;
04817
04818         bool operator == ( TestCase const& other ) const;
04819         bool operator < ( TestCase const& other ) const;
04820
04821     private:
04822         std::shared_ptr<ITestInvoker> test;
04823     };
04824
04825     TestCase makeTestCase( ITestInvoker* testCase,
04826                           std::string const& className,
04827                           NameAndTags const& nameAndTags,
04828                           SourceLineInfo const& lineInfo );
04829 }
04830
04831 #ifndef __clang__
04832 #pragma clang diagnostic pop
04833 #endif
04834
04835 // end catch_test_case_info.h
04836 // start catch_interfaces_runner.h
04837
04838 namespace Catch {
04839
04840     struct IRunner {
04841         virtual ~IRunner();
04842         virtual bool aborting() const = 0;
04843     };
04844 }
04845
04846 // end catch_interfaces_runner.h
04847
04848 #ifndef __OBJC__
04849 // start catch_objc.hpp
04850
04851 #import <objc/runtime.h>
04852
04853 #include <string>
04854
04855 // NB. Any general catch headers included here must be included
04856 // in catch.hpp first to make sure they are included by the single

```

```

04857 // header for non obj-usage
04858
04860 // This protocol is really only here for (self) documenting purposes, since
04861 // all its methods are optional.
04862 @protocol OcFixture
04863
04864 @optional
04865
04866 -(void) setUp;
04867 -(void) tearDown;
04868
04869 @end
04870
04871 namespace Catch {
04872
04873     class OcMethod : public ITestInvoker {
04874     public:
04875         OcMethod( Class cls, SEL sel ) : m_cls( cls ), m_sel( sel ) {}
04876
04877         virtual void invoke() const {
04878             id obj = [[m_cls alloc] init];
04879
04880             performOptionalSelector( obj, @selector(setUp) );
04881             performOptionalSelector( obj, m_sel );
04882             performOptionalSelector( obj, @selector(tearDown) );
04883
04884             arcSafeRelease( obj );
04885         }
04886     private:
04887         virtual ~OcMethod() {}
04888
04889         Class m_cls;
04890         SEL m_sel;
04891     };
04892
04893     namespace Detail{
04894
04895         inline std::string getAnnotation( Class cls,
04896                                         std::string const& annotationName,
04897                                         std::string const& testCaseName ) {
04898             NSString* selStr = [[NSString alloc] initWithFormat:@"Catch_%s_%s",
04899                             annotationName.c_str(), testCaseName.c_str()];
04900             SEL sel = NSSelectorFromString( selStr );
04901             arcSafeRelease( selStr );
04902             id value = performOptionalSelector( cls, sel );
04903             if( value )
04904                 return [(NSString*)value UTF8String];
04905             return "";
04906         }
04907     }
04908
04909     inline std::size_t registerTestMethods() {
04910         std::size_t noTestMethods = 0;
04911         int noClasses = objc_getClassList( nullptr, 0 );
04912
04913         Class* classes = (CATCH_UNSAFE_UNRETAINED Class *)malloc( sizeof(Class) * noClasses);
04914         objc_getClassList( classes, noClasses );
04915
04916         for( int c = 0; c < noClasses; c++ ) {
04917             Class cls = classes[c];
04918             {
04919                 u_int count;
04920                 Method* methods = class_copyMethodList( cls, &count );
04921                 for( u_int m = 0; m < count ; m++ ) {
04922                     SEL selector = method_getName(methods[m]);
04923                     std::string methodName = sel_getName(selector);
04924                     if( startsWith( methodName, "Catch_TestCase_" ) ) {
04925                         std::string testCaseName = methodName.substr( 15 );
04926                         std::string name = Detail::getAnnotation( cls, "Name", testCaseName );
04927                         std::string desc = Detail::getAnnotation( cls, "Description", testCaseName );
04928                         const char* className = class_getName( cls );
04929
04930                         getMutableRegistryHub().registerTest( makeTestCase( new OcMethod( cls,
04931                             selector ), className, NameAndTags( name.c_str(), desc.c_str() ), SourceLineInfo("",0) ) );
04932                         noTestMethods++;
04933                     }
04934                 }
04935                 free(methods);
04936             }
04937             return noTestMethods;
04938         }
04939     }
04940 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
04941
04942     namespace Matchers {

```

```

04943     namespace Impl {
04944     namespace NSStringMatchers {
04945
04946         struct StringHolder : MatcherBase<NSString*>{
04947             StringHolder( NSString* substr ) : m_substr( [substr copy] ){}
04948             StringHolder( NSString const& other ) : m_substr( [other.m_substr copy] ){}
04949             StringHolder() {
04950                 arcSafeRelease( m_substr );
04951             }
04952
04953             bool match( NSString* str ) const override {
04954                 return false;
04955             }
04956
04957             NSString* CATCH_ARC_STRONG m_substr;
04958         };
04959
04960         struct Equals : StringHolder {
04961             Equals( NSString* substr ) : StringHolder( substr ){}
04962
04963             bool match( NSString* str ) const override {
04964                 return (str != nil || m_substr == nil ) &&
04965                     [str isEqualToString:m_substr];
04966             }
04967
04968             std::string describe() const override {
04969                 return "equals string: " + Catch::Detail::stringify( m_substr );
04970             }
04971         };
04972
04973         struct Contains : StringHolder {
04974             Contains( NSString* substr ) : StringHolder( substr ){}
04975
04976             bool match( NSString* str ) const override {
04977                 return (str != nil || m_substr == nil ) &&
04978                     [str rangeOfString:m_substr].location != NSNotFound;
04979             }
04980
04981             std::string describe() const override {
04982                 return "contains string: " + Catch::Detail::stringify( m_substr );
04983             }
04984         };
04985
04986         struct StartsWith : StringHolder {
04987             StartsWith( NSString* substr ) : StringHolder( substr ){}
04988
04989             bool match( NSString* str ) const override {
04990                 return (str != nil || m_substr == nil ) &&
04991                     [str rangeOfString:m_substr].location == 0;
04992             }
04993
04994             std::string describe() const override {
04995                 return "starts with: " + Catch::Detail::stringify( m_substr );
04996             }
04997         };
04998         struct EndsWith : StringHolder {
04999             EndsWith( NSString* substr ) : StringHolder( substr ){}
05000
05001             bool match( NSString* str ) const override {
05002                 return (str != nil || m_substr == nil ) &&
05003                     [str rangeOfString:m_substr].location == [str length] - [m_substr length];
05004             }
05005
05006             std::string describe() const override {
05007                 return "ends with: " + Catch::Detail::stringify( m_substr );
05008             }
05009         };
05010
05011     } // namespace NSStringMatchers
05012 } // namespace Impl
05013
05014 inline Impl::NSStringMatchers::Equals
05015     Equals( NSString* substr ){ return Impl::NSStringMatchers::Equals( substr ); }
05016
05017 inline Impl::NSStringMatchers::Contains
05018     Contains( NSString* substr ){ return Impl::NSStringMatchers::Contains( substr ); }
05019
05020 inline Impl::NSStringMatchers::StartsWith
05021     StartsWith( NSString* substr ){ return Impl::NSStringMatchers::StartsWith( substr ); }
05022
05023 inline Impl::NSStringMatchers::EndsWith
05024     EndsWith( NSString* substr ){ return Impl::NSStringMatchers::EndsWith( substr ); }
05025
05026 } // namespace Matchers
05027
05028 using namespace Matchers;
05029

```

```

05030 #endif // CATCH_CONFIG_DISABLE_MATCHERS
05031
05032 } // namespace Catch
05033
05035 #define OC_MAKE_UNIQUE_NAME( root, uniqueSuffix ) root##uniqueSuffix
05036 #define OC_TEST_CASE2( name, desc, uniqueSuffix ) \
05037 +(NSString*) OC_MAKE_UNIQUE_NAME( Catch_Name_test_, uniqueSuffix ) \
05038 { \
05039     return @ name; \
05040 } \
05041 +(NSString*) OC_MAKE_UNIQUE_NAME( Catch_Description_test_, uniqueSuffix ) \
05042 { \
05043     return @ desc; \
05044 } \
05045 -(void) OC_MAKE_UNIQUE_NAME( Catch_TestCase_test_, uniqueSuffix )
05046
05047 #define OC_TEST_CASE( name, desc ) OC_TEST_CASE2( name, desc, __LINE__ )
05048
05049 // end catch_objc.hpp
05050 #endif
05051
05052 // Benchmarking needs the externally-facing parts of reporters to work
05053 #if defined(CATCH_CONFIG_EXTERNAL_INTERFACES) || defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
05054 // start catch_external_interfaces.h
05055
05056 // start catch_reporter_bases.hpp
05057
05058 // start catch_interfaces_reporter.h
05059
05060 // start catch_config.hpp
05061
05062 // start catch_test_spec_parser.h
05063
05064 #ifdef __clang__
05065 #pragma clang diagnostic push
05066 #pragma clang diagnostic ignored "-Wpadded"
05067 #endif
05068
05069 // start catch_test_spec.h
05070
05071 #ifdef __clang__
05072 #pragma clang diagnostic push
05073 #pragma clang diagnostic ignored "-Wpadded"
05074 #endif
05075
05076 // start catch_wildcard_pattern.h
05077
05078 namespace Catch {
05079 {
05080     class WildcardPattern {
05081     public:
05082         enum WildcardPosition {
05083             NoWildcard = 0,
05084             WildcardAtStart = 1,
05085             WildcardAtEnd = 2,
05086             WildcardAtBothEnds = WildcardAtStart | WildcardAtEnd
05087         };
05088
05089         WildcardPattern( std::string const& pattern, CaseSensitive::Choice caseSensitivity );
05090         virtual ~WildcardPattern() = default;
05091         virtual bool matches( std::string const& str ) const;
05092
05093     private:
05094         std::string normaliseString( std::string const& str ) const;
05095         CaseSensitive::Choice m_caseSensitivity;
05096         WildcardPosition m_wildcard = NoWildcard;
05097         std::string m_pattern;
05098     };
05099 }
05100 }
05101
05102 // end catch_wildcard_pattern.h
05103 #include <string>
05104 #include <vector>
05105 #include <memory>
05106
05107 namespace Catch {
05108     struct IConfig;
05109
05110     class TestSpec {
05111     public:
05112         explicit TestSpec( std::string const& name );
05113         virtual ~TestSpec();
05114         virtual bool matches( TestCaseInfo const& testCase ) const = 0;
05115         std::string const& name() const;
05116     };

```

```

05118     private:
05119         std::string const m_name;
05120     };
05121     using PatternPtr = std::shared_ptr<Pattern>;
05122
05123     class NamePattern : public Pattern {
05124     public:
05125         explicit NamePattern( std::string const& name, std::string const& filterString );
05126         bool matches( TestCaseInfo const& testCase ) const override;
05127     private:
05128         WildcardPattern m_wildcardPattern;
05129     };
05130
05131     class TagPattern : public Pattern {
05132     public:
05133         explicit TagPattern( std::string const& tag, std::string const& filterString );
05134         bool matches( TestCaseInfo const& testCase ) const override;
05135     private:
05136         std::string m_tag;
05137     };
05138
05139     class ExcludedPattern : public Pattern {
05140     public:
05141         explicit ExcludedPattern( PatternPtr const& underlyingPattern );
05142         bool matches( TestCaseInfo const& testCase ) const override;
05143     private:
05144         PatternPtr m_underlyingPattern;
05145     };
05146
05147     struct Filter {
05148         std::vector<PatternPtr> m_patterns;
05149
05150         bool matches( TestCaseInfo const& testCase ) const;
05151         std::string name() const;
05152     };
05153
05154     public:
05155         struct FilterMatch {
05156             std::string name;
05157             std::vector<TestCase const*> tests;
05158         };
05159         using Matches = std::vector<FilterMatch>;
05160         using vectorStrings = std::vector<std::string>;
05161
05162         bool hasFilters() const;
05163         bool matches( TestCaseInfo const& testCase ) const;
05164         Matches matchesByFilter( std::vector<TestCase> const& testCases, IConfig const& config )
05165     const;
05166         const vectorStrings & getInvalidArgs() const;
05167     private:
05168         std::vector<Filter> m_filters;
05169         std::vector<std::string> m_invalidArgs;
05170         friend class TestSpecParser;
05171     };
05172 }
05173
05174 #ifdef __clang__
05175 #pragma clang diagnostic pop
05176 #endif
05177
05178 // end catch_test_spec.h
05179 // start catch_interfaces_tag_alias_registry.h
05180
05181 #include <string>
05182
05183 namespace Catch {
05184
05185     struct TagAlias;
05186
05187     struct ITagAliasRegistry {
05188         virtual ~ITagAliasRegistry();
05189         // Nullptr if not present
05190         virtual TagAlias const* find( std::string const& alias ) const = 0;
05191         virtual std::string expandAliases( std::string const& unexpandedTestSpec ) const = 0;
05192
05193         static ITagAliasRegistry const& get();
05194     };
05195
05196 } // end namespace Catch
05197
05198 // end catch_interfaces_tag_alias_registry.h
05199 namespace Catch {
05200
05201     class TestSpecParser {
05202     enum Mode{ None, Name, QuotedName, Tag, EscapedName };
05203         Mode m_mode = None;

```

```

05204     Mode lastMode = None;
05205     bool m_exclusion = false;
05206     std::size_t m_pos = 0;
05207     std::size_t m_realPatternPos = 0;
05208     std::string m_arg;
05209     std::string m_substring;
05210     std::string m_patternName;
05211     std::vector<std::size_t> m_escapeChars;
05212     TestSpec::Filter m_currentFilter;
05213     TestSpec m_testSpec;
05214     ITagAliasRegistry const* m_tagAliases = nullptr;
05215
05216 public:
05217     TestSpecParser( ITagAliasRegistry const& tagAliases );
05218
05219     TestSpecParser& parse( std::string const& arg );
05220     TestSpec testSpec();
05221
05222 private:
05223     bool visitChar( char c );
05224     void startNewMode( Mode mode );
05225     bool processNoneChar( char c );
05226     void processNameChar( char c );
05227     bool processOtherChar( char c );
05228     void endMode();
05229     void escape();
05230     bool isControlChar( char c ) const;
05231     void saveLastMode();
05232     void revertBackToLastMode();
05233     void addFilter();
05234     bool separate();
05235
05236     // Handles common preprocessing of the pattern for name/tag patterns
05237     std::string preprocessPattern();
05238     // Adds the current pattern as a test name
05239     void addNamePattern();
05240     // Adds the current pattern as a tag
05241     void addTagPattern();
05242
05243     inline void addCharToPattern(char c) {
05244         m_substring += c;
05245         m_patternName += c;
05246         m_realPatternPos++;
05247     }
05248
05249 };
05250 TestSpec parseTestSpec( std::string const& arg );
05251
05252 } // namespace Catch
05253
05254 #ifdef __clang__
05255 #pragma clang diagnostic pop
05256 #endif
05257
05258 // end catch_test_spec_parser.h
05259 // Libstdc++ doesn't like incomplete classes for unique_ptr
05260
05261 #include <memory>
05262 #include <vector>
05263 #include <string>
05264
05265 #ifndef CATCH_CONFIG_CONSOLE_WIDTH
05266 #define CATCH_CONFIG_CONSOLE_WIDTH 80
05267 #endif
05268
05269 namespace Catch {
05270
05271     struct IStream;
05272
05273     struct ConfigData {
05274         bool listTests = false;
05275         bool listTags = false;
05276         bool listReporters = false;
05277         bool listTestNamesOnly = false;
05278
05279         bool showSuccessfulTests = false;
05280         bool shouldDebugBreak = false;
05281         bool noThrow = false;
05282         bool showHelp = false;
05283         bool showInvisibles = false;
05284         bool filenamesAsTags = false;
05285         bool libIdentify = false;
05286
05287         int abortAfter = -1;
05288         unsigned int rngSeed = 0;
05289
05290         bool benchmarkNoAnalysis = false;

```

```

05291     unsigned int benchmarkSamples = 100;
05292     double benchmarkConfidenceInterval = 0.95;
05293     unsigned int benchmarkResamples = 100000;
05294     std::chrono::milliseconds::rep benchmarkWarmupTime = 100;
05295
05296     Verbosity verbosity = Verbosity::Normal;
05297     WarnAbout::What warnings = WarnAbout::Nothing;
05298     ShowDurations::OrNot showDurations = ShowDurations::DefaultForReporter;
05299     double minDuration = -1;
05300     RunTests::InWhatOrder runOrder = RunTests::InDeclarationOrder;
05301     UseColour::YesOrNo useColour = UseColour::Auto;
05302     WaitForKeypress::When waitForKeypress = WaitForKeypress::Never;
05303
05304     std::string outputFilename;
05305     std::string name;
05306     std::string processName;
05307 #ifndef CATCH_CONFIG_DEFAULT_REPORTER
05308 #define CATCH_CONFIG_DEFAULT_REPORTER "console"
05309 #endif
05310     std::string reporterName = CATCH_CONFIG_DEFAULT_REPORTER;
05311 #undef CATCH_CONFIG_DEFAULT_REPORTER
05312
05313     std::vector<std::string> testsOrTags;
05314     std::vector<std::string> sectionsToRun;
05315 };
05316
05317 class Config : public IConfig {
05318 public:
05319
05320     Config() = default;
05321     Config( ConfigData const& data );
05322     virtual ~Config() = default;
05323
05324     std::string const& getFilename() const;
05325
05326     bool listTests() const;
05327     bool listTestNamesOnly() const;
05328     bool listTags() const;
05329     bool listReporters() const;
05330
05331     std::string getProcessName() const;
05332     std::string const& getReporterName() const;
05333
05334     std::vector<std::string> const& getTestsOrTags() const override;
05335     std::vector<std::string> const& getSectionsToRun() const override;
05336
05337     TestSpec const& testSpec() const override;
05338     bool hasTestFilters() const override;
05339
05340     bool showHelp() const;
05341
05342     // IConfig interface
05343     bool allowThrows() const override;
05344     std::ostream& stream() const override;
05345     std::string name() const override;
05346     bool includeSuccessfulResults() const override;
05347     bool warnAboutMissingAssertions() const override;
05348     bool warnAboutNoTests() const override;
05349     ShowDurations::OrNot showDurations() const override;
05350     double minDuration() const override;
05351     RunTests::InWhatOrder runOrder() const override;
05352     unsigned int rngSeed() const override;
05353     UseColour::YesOrNo useColour() const override;
05354     bool shouldDebugBreak() const override;
05355     int abortAfter() const override;
05356     bool showInvisibles() const override;
05357     Verbosity verbosity() const override;
05358     bool benchmarkNoAnalysis() const override;
05359     int benchmarkSamples() const override;
05360     double benchmarkConfidenceInterval() const override;
05361     unsigned int benchmarkResamples() const override;
05362     std::chrono::milliseconds benchmarkWarmupTime() const override;
05363
05364 private:
05365     IStream const* openStream();
05366     ConfigData m_data;
05367
05368     std::unique_ptr<IStream const> m_stream;
05369     TestSpec m_testSpec;
05370     bool m_hasTestFilters = false;
05371 };
05372
05373 } // end namespace Catch
05374
05375 // end catch_config.hpp
05376 // start catch_assertionresult.h

```



```

05378
05379 #include <string>
05380
05381 namespace Catch {
05382
05383     struct AssertionResultData
05384     {
05385         AssertionResultData() = delete;
05386
05387         AssertionResultData( ResultWas::OfType _resultType, LazyExpression const& _lazyExpression );
05388
05389         std::string message;
05390         mutable std::string reconstructedExpression;
05391         LazyExpression lazyExpression;
05392         ResultWas::OfType resultType;
05393
05394         std::string reconstructExpression() const;
05395     };
05396
05397     class AssertionResult {
05398     public:
05399         AssertionResult() = delete;
05400         AssertionResult( AssertionInfo const& info, AssertionResultData const& data );
05401
05402         bool isOk() const;
05403         bool succeeded() const;
05404         ResultWas::OfType getResultType() const;
05405         bool hasExpression() const;
05406         bool hasMessage() const;
05407         std::string getExpression() const;
05408         std::string getExpressionInMacro() const;
05409         bool hasExpandedExpression() const;
05410         std::string getExpandedExpression() const;
05411         std::string getMessage() const;
05412         SourceLineInfo getSourceInfo() const;
05413         StringRef getTestMacroName() const;
05414
05415         //protected:
05416         AssertionInfo m_info;
05417         AssertionResultData m_resultData;
05418     };
05419
05420 } // end namespace Catch
05421
05422 // end catch_assertionresult.h
05423 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
05424 // start catch_estimate.hpp
05425
05426 // Statistics estimates
05427
05428 namespace Catch {
05429     namespace Benchmark {
05430         template <typename Duration>
05431         struct Estimate {
05432             Duration point;
05433             Duration lower_bound;
05434             Duration upper_bound;
05435             double confidence_interval;
05436
05437             template <typename Duration2>
05438             operator Estimate<Duration2>() const {
05439                 return { point, lower_bound, upper_bound, confidence_interval };
05440             }
05441         };
05442     } // namespace Benchmark
05443 } // namespace Catch
05444
05445 // end catch_estimate.hpp
05446 // start catch_outlier_classification.hpp
05447
05448 // Outlier information
05449 namespace Catch {
05450     namespace Benchmark {
05451         struct OutlierClassification {
05452             int samples_seen = 0;
05453             int low_severe = 0; // more than 3 times IQR below Q1
05454             int low_mild = 0; // 1.5 to 3 times IQR below Q1
05455             int high_mild = 0; // 1.5 to 3 times IQR above Q3
05456             int high_severe = 0; // more than 3 times IQR above Q3
05457
05458             int total() const {
05459                 return low_severe + low_mild + high_mild + high_severe;
05460             }
05461         };
05462     } // namespace Benchmark
05463 } // namespace Catch

```

```

05465 } // namespace Catch
05466
05467 // end catch_outlier_classification.hpp
05468
05469 #include <iterator>
05470 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
05471
05472 #include <string>
05473 #include <iosfwd>
05474 #include <map>
05475 #include <set>
05476 #include <memory>
05477 #include <algorithm>
05478
05479 namespace Catch {
05480
05481     struct ReporterConfig {
05482         explicit ReporterConfig( IConfigPtr const& _fullConfig );
05483
05484         ReporterConfig( IConfigPtr const& _fullConfig, std::ostream& _stream );
05485
05486         std::ostream& stream() const;
05487         IConfigPtr fullConfig() const;
05488
05489     private:
05490         std::ostream* m_stream;
05491         IConfigPtr m_fullConfig;
05492     };
05493
05494     struct ReporterPreferences {
05495         bool shouldRedirectStdOut = false;
05496         bool shouldReportAllAssertions = false;
05497     };
05498
05499     template<typename T>
05500     struct LazyStat : Option<T> {
05501         LazyStat& operator=( T const& _value ) {
05502             Option<T>::operator=( _value );
05503             used = false;
05504             return *this;
05505         }
05506         void reset() {
05507             Option<T>::reset();
05508             used = false;
05509         }
05510         bool used = false;
05511     };
05512
05513     struct TestRunInfo {
05514         TestRunInfo( std::string const& _name );
05515         std::string name;
05516     };
05517
05518     struct GroupInfo {
05519         GroupInfo( std::string const& _name,
05520                   std::size_t _groupIndex,
05521                   std::size_t _groupsCount );
05522
05523         std::string name;
05524         std::size_t groupIndex;
05525         std::size_t groupsCounts;
05526     };
05527
05528     struct AssertionStats {
05529         AssertionStats( AssertionResult const& _assertionResult,
05530                         std::vector<MessageInfo> const& _infoMessages,
05531                         Totals const& _totals );
05532
05533         AssertionStats( AssertionStats const& ) = default;
05534         AssertionStats( AssertionStats && ) = default;
05535         AssertionStats& operator = ( AssertionStats const& ) = delete;
05536         AssertionStats& operator = ( AssertionStats && ) = delete;
05537         virtual ~AssertionStats();
05538
05539         AssertionResult assertionResult;
05540         std::vector<MessageInfo> infoMessages;
05541         Totals totals;
05542     };
05543
05544     struct SectionStats {
05545         SectionStats( SectionInfo const& _sectionInfo,
05546                       Counts const& _assertions,
05547                       double _durationInSeconds,
05548                       bool _missingAssertions );
05549         SectionStats( SectionStats const& ) = default;
05550         SectionStats( SectionStats && ) = default;
05551         SectionStats& operator = ( SectionStats const& ) = default;
05552         SectionStats& operator = ( SectionStats && ) = default;

```

```

05552     virtual ~SectionStats();
05553
05554     SectionInfo sectionInfo;
05555     Counts assertions;
05556     double durationInSeconds;
05557     bool missingAssertions;
05558 };
05559
05560 struct TestCaseStats {
05561     TestCaseStats( TestCaseInfo const& _testInfo,
05562                   Totals const& _totals,
05563                   std::string const& _stdOut,
05564                   std::string const& _stdErr,
05565                   bool _aborting );
05566
05567     TestCaseStats( TestCaseStats const& )           = default;
05568     TestCaseStats( TestCaseStats && )             = default;
05569     TestCaseStats& operator = ( TestCaseStats const& ) = default;
05570     TestCaseStats& operator = ( TestCaseStats && )   = default;
05571     virtual ~TestCaseStats();
05572
05573     TestCaseInfo testInfo;
05574     Totals totals;
05575     std::string stdOut;
05576     std::string stdErr;
05577     bool aborting;
05578 };
05579
05580 struct TestGroupStats {
05581     TestGroupStats( GroupInfo const& _groupInfo,
05582                   Totals const& _totals,
05583                   bool _aborting );
05584     TestGroupStats( GroupInfo const& _groupInfo );
05585
05586     TestGroupStats( TestGroupStats const& )           = default;
05587     TestGroupStats( TestGroupStats && )             = default;
05588     TestGroupStats& operator = ( TestGroupStats const& ) = default;
05589     TestGroupStats& operator = ( TestGroupStats && )   = default;
05590     virtual ~TestGroupStats();
05591
05592     GroupInfo groupInfo;
05593     Totals totals;
05594     bool aborting;
05595 };
05596
05597 struct TestRunStats {
05598     TestRunStats( TestRunInfo const& _runInfo,
05599                 Totals const& _totals,
05600                 bool _aborting );
05601
05602     TestRunStats( TestRunStats const& )           = default;
05603     TestRunStats( TestRunStats && )             = default;
05604     TestRunStats& operator = ( TestRunStats const& ) = default;
05605     TestRunStats& operator = ( TestRunStats && )   = default;
05606     virtual ~TestRunStats();
05607
05608     TestRunInfo runInfo;
05609     Totals totals;
05610     bool aborting;
05611 };
05612
05613 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
05614 struct BenchmarkInfo {
05615     std::string name;
05616     double estimatedDuration;
05617     int iterations;
05618     int samples;
05619     unsigned int resamples;
05620     double clockResolution;
05621     double clockCost;
05622 };
05623
05624 template <class Duration>
05625 struct BenchmarkStats {
05626     BenchmarkInfo info;
05627
05628     std::vector<Duration> samples;
05629     Benchmark::Estimate<Duration> mean;
05630     Benchmark::Estimate<Duration> standardDeviation;
05631     Benchmark::OutlierClassification outliers;
05632     double outlierVariance;
05633
05634     template <typename Duration2>
05635     operator BenchmarkStats<Duration2>() const {
05636         std::vector<Duration2> samples2;
05637         samples2.reserve(samples.size());
05638         std::transform(samples.begin(), samples.end(), std::back_inserter(samples2), [](Duration

```

```

    d) { return Duration2(d); });
05639         return {
05640             info,
05641             std::move(samples2),
05642             mean,
05643             standardDeviation,
05644             outliers,
05645             outlierVariance,
05646         };
05647     }
05648 };
05649 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
05650
05651 struct IStreamingReporter {
05652     virtual ~IStreamingReporter() = default;
05653
05654     // Implementing class must also provide the following static methods:
05655     // static std::string getDescription();
05656     // static std::set<Verbosity> getSupportedVerbsities()
05657
05658     virtual ReporterPreferences getPreferences() const = 0;
05659
05660     virtual void noMatchingTestCases( std::string const& spec ) = 0;
05661
05662     virtual void reportInvalidArguments(std::string const&) {}
05663
05664     virtual void testRunStarting( TestRunInfo const& testRunInfo ) = 0;
05665     virtual void testGroupStarting( GroupInfo const& groupInfo ) = 0;
05666
05667     virtual void testCaseStarting( TestCaseInfo const& testInfo ) = 0;
05668     virtual void sectionStarting( SectionInfo const& sectionInfo ) = 0;
05669
05670 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
05671     virtual void benchmarkPreparing( std::string const& ) {}
05672     virtual void benchmarkStarting( BenchmarkInfo const& ) {}
05673     virtual void benchmarkEnded( BenchmarkStats<> const& ) {}
05674     virtual void benchmarkFailed( std::string const& ) {}
05675 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
05676
05677     virtual void assertionStarting( AssertionInfo const& assertionInfo ) = 0;
05678
05679     // The return value indicates if the messages buffer should be cleared:
05680     virtual bool assertionEnded( AssertionStats const& assertionStats ) = 0;
05681
05682     virtual void sectionEnded( SectionStats const& sectionStats ) = 0;
05683     virtual void testCaseEnded( TestCaseStats const& testCaseStats ) = 0;
05684     virtual void testGroupEnded( TestGroupStats const& testGroupStats ) = 0;
05685     virtual void testRunEnded( TestRunStats const& testRunStats ) = 0;
05686
05687     virtual void skipTest( TestCaseInfo const& testInfo ) = 0;
05688
05689     // Default empty implementation provided
05690     virtual void fatalErrorEncountered( StringRef name );
05691
05692     virtual bool isMulti() const;
05693 };
05694 using IStreamingReporterPtr = std::unique_ptr<IStreamingReporter>;
05695
05696 struct IReporterFactory {
05697     virtual ~IReporterFactory();
05698     virtual IStreamingReporterPtr create( ReporterConfig const& config ) const = 0;
05699     virtual std::string getDescription() const = 0;
05700 };
05701 using IReporterFactoryPtr = std::shared_ptr<IReporterFactory>;
05702
05703 struct IReporterRegistry {
05704     using FactoryMap = std::map<std::string, IReporterFactoryPtr>;
05705     using Listeners = std::vector<IReporterFactoryPtr>;
05706
05707     virtual ~IReporterRegistry();
05708     virtual IStreamingReporterPtr create( std::string const& name, IConfigPtr const& config )
05709     const = 0;
05710     virtual FactoryMap const& getFactories() const = 0;
05711     virtual Listeners const& getListeners() const = 0;
05712 };
05713 } // end namespace Catch
05714 // end catch_interfaces_reporter.h
05715 #include <algorithm>
05716 #include <cstring>
05717 #include <cstdio>
05718 #include <memory>
05719 #include <ostream>
05720 #include <cassert>
05721 #include <memory>
05722 #include <ostream>
05723

```

```

05724 namespace Catch {
05725     void prepareExpandedExpression( AssertionResult& result );
05726
05727     // Returns double formatted as %.3f (format expected on output)
05728     std::string getFormattedDuration( double duration );
05729
05730     bool shouldShowDuration( IConfig const& config, double duration );
05731
05732     std::string serializeFilters( std::vector<std::string> const& container );
05733
05734     template<typename DerivedT>
05735     struct StreamingReporterBase : IStreamingReporter {
05736         StreamingReporterBase( ReporterConfig const& _config )
05737             : m_config( _config.fullConfig() ),
05738               stream( _config.stream() ) {
05739             {
05740                 m_reporterPrefs.shouldRedirectStdOut = false;
05741                 if( !DerivedT::getSupportedVerbsosities().count( m_config->verbosity() ) )
05742                     CATCH_ERROR( "Verbosity level not supported by this reporter" );
05743             }
05744
05745             ReporterPreferences getPreferences() const override {
05746                 return m_reporterPrefs;
05747             }
05748
05749             static std::set<Verbosity> getSupportedVerbsosities() {
05750                 return { Verbosity::Normal };
05751             }
05752
05753             ~StreamingReporterBase() override = default;
05754
05755             void noMatchingTestCases( std::string const& ) override {}
05756
05757             void reportInvalidArguments( std::string const& ) override {}
05758
05759             void testRunStarting( TestRunInfo const& _testRunInfo ) override {
05760                 currentTestRunInfo = _testRunInfo;
05761             }
05762
05763             void testGroupStarting( GroupInfo const& _groupInfo ) override {
05764                 currentGroupInfo = _groupInfo;
05765             }
05766
05767             void testCaseStarting( TestCaseInfo const& _testInfo ) override {
05768                 currentTestCaseInfo = _testInfo;
05769             }
05770
05771             void sectionStarting( SectionInfo const& _sectionInfo ) override {
05772                 m_sectionStack.push_back( _sectionInfo );
05773             }
05774
05775             void sectionEnded( SectionStats const& /* _sectionStats */ ) override {
05776                 m_sectionStack.pop_back();
05777             }
05778
05779             void testCaseEnded( TestCaseStats const& /* _testCaseStats */ ) override {
05780                 currentTestCaseInfo.reset();
05781             }
05782
05783             void testGroupEnded( TestGroupStats const& /* _testGroupStats */ ) override {
05784                 currentGroupInfo.reset();
05785             }
05786
05787             void testRunEnded( TestRunStats const& /* _testRunStats */ ) override {
05788                 currentTestCaseInfo.reset();
05789                 currentGroupInfo.reset();
05790                 currentTestRunInfo.reset();
05791             }
05792
05793             void skipTest( TestCaseInfo const& ) override {
05794                 // Don't do anything with this by default.
05795                 // It can optionally be overridden in the derived class.
05796             }
05797
05798             IConfigPtr m_config;
05799             std::ostream& stream;
05800
05801             LazyStat<TestRunInfo> currentTestRunInfo;
05802             LazyStat<GroupInfo> currentGroupInfo;
05803             LazyStat<TestCaseInfo> currentTestCaseInfo;
05804
05805             std::vector<SectionInfo> m_sectionStack;
05806             ReporterPreferences m_reporterPrefs;
05807         };
05808
05809         template<typename DerivedT>
05810         struct CumulativeReporterBase : IStreamingReporter {
05811             template<typename T, typename ChildNodeT>
05812             struct Node {
05813                 explicit Node( T const& _value ) : value( _value ) {}

```

```

05812         virtual ~Node() {}
05813
05814         using ChildNodes = std::vector<std::shared_ptr<ChildNodeT>;
05815         T value;
05816         ChildNodes children;
05817     };
05818     struct SectionNode {
05819         explicit SectionNode(SectionStats const& _stats) : stats(_stats) {}
05820         virtual ~SectionNode() = default;
05821
05822         bool operator == (SectionNode const& other) const {
05823             return stats.sectionInfo.lineInfo == other.stats.sectionInfo.lineInfo;
05824         }
05825         bool operator == (std::shared_ptr<SectionNode> const& other) const {
05826             return operator==( *other );
05827         }
05828
05829         SectionStats stats;
05830         using ChildSections = std::vector<std::shared_ptr<SectionNode>;
05831         using Assertions = std::vector<AssertionStats>;
05832         ChildSections childSections;
05833         Assertions assertions;
05834         std::string stdOut;
05835         std::string stdErr;
05836     };
05837
05838     struct BySectionInfo {
05839         BySectionInfo( SectionInfo const& other ) : m_other( other ) {}
05840         BySectionInfo( BySectionInfo const& other ) : m_other( other.m_other ) {}
05841         bool operator() (std::shared_ptr<SectionNode> const& node) const {
05842             return ((node->stats.sectionInfo.name == m_other.name) &&
05843                 (node->stats.sectionInfo.lineInfo == m_other.lineInfo));
05844         }
05845         void operator=(BySectionInfo const&) = delete;
05846
05847     private:
05848         SectionInfo const& m_other;
05849     };
05850
05851     using TestCaseNode = Node<TestCaseStats, SectionNode>;
05852     using TestGroupNode = Node<TestGroupStats, TestCaseNode>;
05853     using TestRunNode = Node<TestRunStats, TestGroupNode>;
05854
05855     CumulativeReporterBase( ReporterConfig const& _config )
05856     :   m_config( _config.fullConfig() ),
05857         stream( _config.stream() )
05858     {
05859         m_reporterPrefs.shouldRedirectStdOut = false;
05860         if( !DerivedT::getSupportedVerbsities().count( m_config->verbosity() ) )
05861             CATCH_ERROR( "Verbosity level not supported by this reporter" );
05862     }
05863     ~CumulativeReporterBase() override = default;
05864
05865     ReporterPreferences getPreferences() const override {
05866         return m_reporterPrefs;
05867     }
05868
05869     static std::set<Verbosity> getSupportedVerbsities() {
05870         return { Verbosity::Normal };
05871     }
05872
05873     void testRunStarting( TestRunInfo const& ) override {}
05874     void testGroupStarting( GroupInfo const& ) override {}
05875
05876     void testCaseStarting( TestCaseInfo const& ) override {}
05877
05878     void sectionStarting( SectionInfo const& sectionInfo ) override {
05879         SectionStats incompleteStats( sectionInfo, Counts(), 0, false );
05880         std::shared_ptr<SectionNode> node;
05881         if( m_sectionStack.empty() ) {
05882             if( !m_rootSection )
05883                 m_rootSection = std::make_shared<SectionNode>( incompleteStats );
05884             node = m_rootSection;
05885         }
05886         else {
05887             SectionNode& parentNode = *m_sectionStack.back();
05888             auto it =
05889                 std::find_if( parentNode.childSections.begin(),
05890                             parentNode.childSections.end(),
05891                             BySectionInfo( sectionInfo ) );
05892             if( it == parentNode.childSections.end() ) {
05893                 node = std::make_shared<SectionNode>( incompleteStats );
05894                 parentNode.childSections.push_back( node );
05895             }
05896             else
05897                 node = *it;
05898         }
05899     }

```

```

05899         m_sectionStack.push_back( node );
05900         m_deepestSection = std::move(node);
05901     }
05902
05903     void assertionStarting(AssertionInfo const& override) {}
05904
05905     bool assertionEnded(AssertionStats const& assertionStats) override {
05906         assert(!m_sectionStack.empty());
05907         // AssertionResult holds a pointer to a temporary DecomposedExpression,
05908         // which getExpandedExpression() calls to build the expression string.
05909         // Our section stack copy of the assertionResult will likely outlive the
05910         // temporary, so it must be expanded or discarded now to avoid calling
05911         // a destroyed object later.
05912         prepareExpandedExpression(const_cast<AssertionResult*>( assertionStats.assertionResult )
05913 );
05914
05915         SectionNode& sectionNode = *m_sectionStack.back();
05916         sectionNode.assertions.push_back(assertionStats);
05917         return true;
05918     }
05919
05920     void sectionEnded(SectionStats const& sectionStats) override {
05921         assert(!m_sectionStack.empty());
05922         SectionNode& node = *m_sectionStack.back();
05923         node.stats = sectionStats;
05924         m_sectionStack.pop_back();
05925     }
05926
05927     void testCaseEnded(TestCaseStats const& testCaseStats) override {
05928         auto node = std::make_shared<TestCaseNode>(testCaseStats);
05929         assert(m_sectionStack.size() == 0);
05930         node->children.push_back(m_rootSection);
05931         m_testCases.push_back(node);
05932         m_rootSection.reset();
05933
05934         assert(m_deepestSection);
05935         m_deepestSection->stdOut = testCaseStats.stdOut;
05936         m_deepestSection->stdErr = testCaseStats.stdErr;
05937     }
05938
05939     void testGroupEnded(TestGroupStats const& testGroupStats) override {
05940         auto node = std::make_shared<TestGroupNode>(testGroupStats);
05941         node->children.swap(m_testCases);
05942         m_testGroups.push_back(node);
05943     }
05944
05945     void testRunEnded(TestRunStats const& testRunStats) override {
05946         auto node = std::make_shared<TestRunNode>(testRunStats);
05947         node->children.swap(m_testGroups);
05948         m_testRuns.push_back(node);
05949         testRunEndedCumulative();
05950     }
05951
05952     virtual void testRunEndedCumulative() = 0;
05953
05954     void skipTest(TestCaseInfo const& override) {}
05955
05956     IConfigPtr m_config;
05957     std::ostream& stream;
05958     std::vector<AssertionStats> m_assertions;
05959     std::vector<std::vector<std::shared_ptr<SectionNode>>> m_sections;
05960     std::vector<std::shared_ptr<TestCaseNode> m_testCases;
05961     std::vector<std::shared_ptr<TestGroupNode> m_testGroups;
05962
05963     std::vector<std::shared_ptr<TestRunNode> m_testRuns;
05964
05965     std::shared_ptr<SectionNode> m_rootSection;
05966     std::shared_ptr<SectionNode> m_deepestSection;
05967     std::vector<std::shared_ptr<SectionNode> m_sectionStack;
05968     ReporterPreferences m_reporterPrefs;
05969 };
05970
05971 template<char C>
05972 char const* getLineOfChars() {
05973     static char line[CATCH_CONFIG_CONSOLE_WIDTH] = {0};
05974     if( !*line ) {
05975         std::memset( line, C, CATCH_CONFIG_CONSOLE_WIDTH-1 );
05976         line[CATCH_CONFIG_CONSOLE_WIDTH-1] = 0;
05977     }
05978     return line;
05979 }
05980
05981 struct TestEventListenerBase : StreamingReporterBase<TestEventListenerBase> {
05982     TestEventListenerBase( ReporterConfig const& _config );
05983
05984     static std::set<Verbosity> getSupportedVerbsosities();
05985
05986     void assertionStarting(AssertionInfo const& override);
05987     bool assertionEnded(AssertionStats const& override);
05988 };
05989 } // end namespace Catch
05990

```

```

05985 // end catch_reporter_bases.hpp
05986 // start catch_console_colour.h
05987
05988 namespace Catch {
05989
05990     struct Colour {
05991         enum Code {
05992             None = 0,
05993
05994             White,
05995             Red,
05996             Green,
05997             Blue,
05998             Cyan,
05999             Yellow,
06000             Grey,
06001
06002             Bright = 0x10,
06003
06004             BrightRed = Bright | Red,
06005             BrightGreen = Bright | Green,
06006             LightGrey = Bright | Grey,
06007             BrightWhite = Bright | White,
06008             BrightYellow = Bright | Yellow,
06009
06010             // By intention
06011             FileName = LightGrey,
06012             Warning = BrightYellow,
06013             ResultError = BrightRed,
06014             ResultSuccess = BrightGreen,
06015             ResultExpectedFailure = Warning,
06016
06017             Error = BrightRed,
06018             Success = Green,
06019
06020             OriginalExpression = Cyan,
06021             ReconstructedExpression = BrightYellow,
06022
06023             SecondaryText = LightGrey,
06024             Headers = White
06025         };
06026
06027         // Use constructed object for RAII guard
06028         Colour( Code _colourCode );
06029         Colour( Colour&& other ) noexcept;
06030         Colour& operator=( Colour&& other ) noexcept;
06031         ~Colour();
06032
06033         // Use static method for one-shot changes
06034         static void use( Code _colourCode );
06035
06036     private:
06037         bool m_moved = false;
06038     };
06039
06040     std::ostream& operator << ( std::ostream& os, Colour const& );
06041
06042 } // end namespace Catch
06043
06044 // end catch_console_colour.h
06045 // start catch_reporter_registrars.hpp
06046
06047 namespace Catch {
06048
06049     template<typename T>
06050     class ReporterRegistrar {
06051     public:
06052         class ReporterFactory : public IReporterFactory {
06053         public:
06054             IStreamingReporterPtr create( ReporterConfig const& config ) const override {
06055                 return std::unique_ptr<T>( new T( config ) );
06056             }
06057
06058             std::string getDescription() const override {
06059                 return T::getDescription();
06060             }
06061         };
06062
06063     public:
06064         explicit ReporterRegistrar( std::string const& name ) {
06065             getMutableRegistryHub().registerReporter( name, std::make_shared<ReporterFactory>() );
06066         }
06067     };
06068
06069     template<typename T>

```



```

06072     class ListenerRegistrar {
06073     public:
06074         class ListenerFactory : public IReporterFactory {
06075         public:
06076             IStreamingReporterPtr create( ReporterConfig const& config ) const override {
06077                 return std::unique_ptr<T>( new T( config ) );
06078             }
06079             std::string getDescription() const override {
06080                 return std::string();
06081             }
06082         };
06083     };
06084 public:
06085     ListenerRegistrar() {
06086         getMutableRegistryHub().registerListener( std::make_shared<ListenerFactory>() );
06087     }
06088 };
06089
06090 }
06091
06092 #if !defined(CATCH_CONFIG_DISABLE)
06093
06094 #define CATCH_REGISTER_REPORTER( name, reporterType ) \
06095     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
06096     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
06097     namespace{ Catch::ReporterRegistrar<reporterType> catch_internal_RegistrarFor##reporterType( name
06098 ); } \
06099     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
06100
06101 #define CATCH_REGISTER_LISTENER( listenerType ) \
06102     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
06103     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
06104     namespace{ Catch::ListenerRegistrar<listenerType> catch_internal_RegistrarFor##listenerType; } \
06105     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
06106 #else // CATCH_CONFIG_DISABLE
06107 #define CATCH_REGISTER_REPORTER(name, reporterType)
06108 #define CATCH_REGISTER_LISTENER(listenerType)
06109 #endif // CATCH_CONFIG_DISABLE
06110
06111 // end catch_reporter_registrars.hpp
06112 // Allow users to base their work off existing reporters
06113 // start catch_reporter_compact.h
06114
06115 namespace Catch {
06116     struct CompactReporter : StreamingReporterBase<CompactReporter> {
06117     public:
06118         using StreamingReporterBase::StreamingReporterBase;
06119         ~CompactReporter() override;
06120         static std::string getDescription();
06121         void noMatchingTestCases(std::string const& spec) override;
06122         void assertionStarting(AssertionInfo const&) override;
06123         bool assertionEnded(AssertionStats const& _assertionStats) override;
06124         void sectionEnded(SectionStats const& _sectionStats) override;
06125         void testRunEnded(TestRunStats const& _testRunStats) override;
06126     };
06127 } // end namespace Catch
06128
06129 // end catch_reporter_compact.h
06130 // start catch_reporter_console.h
06131
06132 #if defined(_MSC_VER)
06133 #pragma warning(push)
06134 #pragma warning(disable:4061) // Not all labels are EXPLICITLY handled in switch
06135 // Note that 4062 (not all labels are handled) is enabled
06136 #endif
06137 #endif
06138
06139 namespace Catch {
06140     // Fwd decls
06141     struct SummaryColumn;
06142     class TablePrinter;
06143
06144     struct ConsoleReporter : StreamingReporterBase<ConsoleReporter> {
06145         std::unique_ptr<TablePrinter> m_tablePrinter;
06146     };

```

```

06158     ConsoleReporter(ReporterConfig const& config);
06159     ~ConsoleReporter() override;
06160     static std::string getDescription();
06161
06162     void noMatchingTestCases(std::string const& spec) override;
06163
06164     void reportInvalidArguments(std::string const& arg) override;
06165
06166     void assertionStarting(AssertionInfo const&) override;
06167
06168     bool assertionEnded(AssertionStats const& _assertionStats) override;
06169
06170     void sectionStarting(SectionInfo const& _sectionInfo) override;
06171     void sectionEnded(SectionStats const& _sectionStats) override;
06172
06173 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
06174     void benchmarkPreparing(std::string const& name) override;
06175     void benchmarkStarting(BenchmarkInfo const& info) override;
06176     void benchmarkEnded(BenchmarkStats<> const& stats) override;
06177     void benchmarkFailed(std::string const& error) override;
06178 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
06179
06180     void testCaseEnded(TestCaseStats const& _testCaseStats) override;
06181     void testGroupEnded(TestGroupStats const& _testGroupStats) override;
06182     void testRunEnded(TestRunStats const& _testRunStats) override;
06183     void testRunStarting(TestRunInfo const& _testRunInfo) override;
06184 private:
06185
06186     void lazyPrint();
06187
06188     void lazyPrintWithoutClosingBenchmarkTable();
06189     void lazyPrintRunInfo();
06190     void lazyPrintGroupInfo();
06191     void printTestCaseAndSectionHeader();
06192
06193     void printClosedHeader(std::string const& _name);
06194     void printOpenHeader(std::string const& _name);
06195
06196     // if string has a : in first line will set indent to follow it on
06197     // subsequent lines
06198     void printHeaderString(std::string const& _string, std::size_t indent = 0);
06199
06200     void printTotals(Totals const& totals);
06201     void printSummaryRow(std::string const& label, std::vector<SummaryColumn> const& cols,
06202                          std::size_t row);
06203
06204     void printTotalsDivider(Totals const& totals);
06205     void printSummaryDivider();
06206     void printTestFilters();
06207 private:
06208     bool m_headerPrinted = false;
06209 };
06210
06211 } // end namespace Catch
06212
06213 #if defined(_MSC_VER)
06214 #pragma warning(pop)
06215 #endif
06216
06217 // end catch_reporter_console.h
06218 // start catch_reporter_junit.h
06219
06220 // start catch_xmlwriter.h
06221
06222 #include <vector>
06223
06224 namespace Catch {
06225     enum class XmlFormatting {
06226         None = 0x00,
06227         Indent = 0x01,
06228         Newline = 0x02,
06229     };
06230
06231     XmlFormatting operator | (XmlFormatting lhs, XmlFormatting rhs);
06232     XmlFormatting operator & (XmlFormatting lhs, XmlFormatting rhs);
06233
06234     class XmlEncode {
06235     public:
06236         enum ForWhat { ForTextNodes, ForAttributes };
06237
06238         XmlEncode( std::string const& str, ForWhat forWhat = ForTextNodes );
06239
06240         void encodeTo( std::ostream& os ) const;
06241
06242         friend std::ostream& operator << ( std::ostream& os, XmlEncode const& xmlEncode );
06243

```

```

06244     private:
06245         std::string m_str;
06246         ForWhat m_forWhat;
06247     };
06248
06249     class XmlWriter {
06250     public:
06251
06252         class ScopedElement {
06253         public:
06254             ScopedElement( XmlWriter* writer, XmlFormatting fmt );
06255
06256             ScopedElement( ScopedElement&& other ) noexcept;
06257             ScopedElement& operator=( ScopedElement&& other ) noexcept;
06258
06259             ~ScopedElement();
06260
06261             ScopedElement& writeText( std::string const& text, XmlFormatting fmt =
06262             XmlFormatting::Newline | XmlFormatting::Indent );
06263
06264             template<typename T>
06265             ScopedElement& writeAttribute( std::string const& name, T const& attribute ) {
06266                 m_writer->writeAttribute( name, attribute );
06267                 return *this;
06268             }
06269
06270         private:
06271             mutable XmlWriter* m_writer = nullptr;
06272             XmlFormatting m_fmt;
06273         };
06274
06275         XmlWriter( std::ostream& os = Catch::cout() );
06276         ~XmlWriter();
06277
06278         XmlWriter( XmlWriter const& ) = delete;
06279         XmlWriter& operator=( XmlWriter const& ) = delete;
06280
06281         XmlWriter& startElement( std::string const& name, XmlFormatting fmt = XmlFormatting::Newline |
06282         XmlFormatting::Indent);
06283
06284         ScopedElement scopedElement( std::string const& name, XmlFormatting fmt =
06285         XmlFormatting::Newline | XmlFormatting::Indent);
06286
06287         XmlWriter& endElement(XmlFormatting fmt = XmlFormatting::Newline | XmlFormatting::Indent);
06288
06289         XmlWriter& writeAttribute( std::string const& name, std::string const& attribute );
06290
06291         XmlWriter& writeAttribute( std::string const& name, bool attribute );
06292
06293         template<typename T>
06294         XmlWriter& writeAttribute( std::string const& name, T const& attribute ) {
06295             ReusableStringStream rss;
06296             rss << attribute;
06297             return writeAttribute( name, rss.str() );
06298         }
06299
06300         XmlWriter& writeText( std::string const& text, XmlFormatting fmt = XmlFormatting::Newline |
06301         XmlFormatting::Indent);
06302
06303         XmlWriter& writeComment(std::string const& text, XmlFormatting fmt = XmlFormatting::Newline |
06304         XmlFormatting::Indent);
06305
06306         void writeStylesheetRef( std::string const& url );
06307
06308         XmlWriter& writeBlankLine();
06309
06310         void ensureTagClosed();
06311
06312     private:
06313
06314         void applyFormatting(XmlFormatting fmt);
06315
06316         void writeDeclaration();
06317
06318         void newlineIfNecessary();
06319
06320         bool m_tagIsOpen = false;
06321         bool m_needsNewline = false;
06322         std::vector<std::string> m_tags;
06323         std::string m_indent;
06324         std::ostream& m_os;
06325     };
06326
06327 }
06328
06329 // end catch_xmlwriter.h
06330 namespace Catch {

```

```

06326
06327     class JunitReporter : public CumulativeReporterBase<JunitReporter> {
06328     public:
06329         JunitReporter(ReporterConfig const& _config);
06330
06331         ~JunitReporter() override;
06332
06333         static std::string getDescription();
06334
06335         void noMatchingTestCases(std::string const& /*spec*/) override;
06336
06337         void testRunStarting(TestRunInfo const& runInfo) override;
06338
06339         void testGroupStarting(GroupInfo const& groupInfo) override;
06340
06341         void testCaseStarting(TestCaseInfo const& testCaseInfo) override;
06342         bool assertionEnded(AssertionStats const& assertionStats) override;
06343
06344         void testCaseEnded(TestCaseStats const& testCaseStats) override;
06345
06346         void testGroupEnded(TestGroupStats const& testGroupStats) override;
06347
06348         void testRunEndedCumulative() override;
06349
06350         void writeGroup(TestGroupNode const& groupNode, double suiteTime);
06351
06352         void writeTestCase(TestCaseNode const& testCaseNode);
06353
06354         void writeSection( std::string const& className,
06355                           std::string const& rootName,
06356                           SectionNode const& sectionNode,
06357                           bool testOkToFail );
06358
06359         void writeAssertions(SectionNode const& sectionNode);
06360         void writeAssertion(AssertionStats const& stats);
06361
06362         XmlWriter xml;
06363         Timer suiteTimer;
06364         std::string stdOutForSuite;
06365         std::string stdErrForSuite;
06366         unsigned int unexpectedExceptions = 0;
06367         bool m_okToFail = false;
06368     };
06369
06370 } // end namespace Catch
06371
06372 // end catch_reporter_junit.h
06373 // start catch_reporter_xml.h
06374
06375 namespace Catch {
06376     class XmlReporter : public StreamingReporterBase<XmlReporter> {
06377     public:
06378         XmlReporter(ReporterConfig const& _config);
06379
06380         ~XmlReporter() override;
06381
06382         static std::string getDescription();
06383
06384         virtual std::string getStylesheetRef() const;
06385
06386         void writeSourceInfo(SourceLineInfo const& sourceInfo);
06387
06388     public: // StreamingReporterBase
06389
06390         void noMatchingTestCases(std::string const& s) override;
06391
06392         void testRunStarting(TestRunInfo const& testInfo) override;
06393
06394         void testGroupStarting(GroupInfo const& groupInfo) override;
06395
06396         void testCaseStarting(TestCaseInfo const& testInfo) override;
06397
06398         void sectionStarting(SectionInfo const& sectionInfo) override;
06399
06400         void assertionStarting(AssertionInfo const&) override;
06401
06402         bool assertionEnded(AssertionStats const& assertionStats) override;
06403
06404         void sectionEnded(SectionStats const& sectionStats) override;
06405
06406         void testCaseEnded(TestCaseStats const& testCaseStats) override;
06407
06408         void testGroupEnded(TestGroupStats const& testGroupStats) override;
06409
06410         void testRunEnded(TestRunStats const& testRunStats) override;
06411
06412         #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)

```

```

06413         void benchmarkPreparing(std::string const& name) override;
06414         void benchmarkStarting(BenchmarkInfo const&) override;
06415         void benchmarkEnded(BenchmarkStats<> const&) override;
06416         void benchmarkFailed(std::string const&) override;
06417 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
06418
06419     private:
06420         Timer m_testCaseTimer;
06421         XmlWriter m_xml;
06422         int m_sectionDepth = 0;
06423     };
06424
06425 } // end namespace Catch
06426
06427 // end catch_reporter_xml.h
06428
06429 // end catch_external_interfaces.h
06430 #endif
06431
06432 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
06433 // start catch_benchmarking_all.hpp
06434
06435 // A proxy header that includes all of the benchmarking headers to allow
06436 // concise include of the benchmarking features. You should prefer the
06437 // individual includes in standard use.
06438
06439 // start catch_benchmark.hpp
06440
06441 // Benchmark
06442
06443 // start catch_chronometer.hpp
06444
06445 // User-facing chronometer
06446
06447
06448 // start catch_clock.hpp
06449
06450 // Clocks
06451
06452
06453 #include <chrono>
06454 #include <ratio>
06455
06456 namespace Catch {
06457     namespace Benchmark {
06458         template <typename Clock>
06459         using ClockDuration = typename Clock::duration;
06460         template <typename Clock>
06461         using FloatDuration = std::chrono::duration<double, typename Clock::period>;
06462
06463         template <typename Clock>
06464         using TimePoint = typename Clock::time_point;
06465
06466         using default_clock = std::chrono::steady_clock;
06467
06468         template <typename Clock>
06469         struct now {
06470             TimePoint<Clock> operator()() const {
06471                 return Clock::now();
06472             }
06473         };
06474
06475         using fp_seconds = std::chrono::duration<double, std::ratio<1>;
06476     } // namespace Benchmark
06477 } // namespace Catch
06478
06479 // end catch_clock.hpp
06480 // start catch_optimizer.hpp
06481
06482 // Hinting the optimizer
06483
06484
06485 #if defined(_MSC_VER)
06486 #    include <atomic> // atomic_thread_fence
06487 #endif
06488
06489 namespace Catch {
06490     namespace Benchmark {
06491         #if defined(__GNUC__) || defined(__clang__)
06492             template <typename T>
06493             inline void keep_memory(T* p) {
06494                 asm volatile("" : : "g"(p) : "memory");
06495             }
06496             inline void keep_memory() {
06497                 asm volatile("" : : : "memory");
06498             }
06499
06500         #endif
06501     }
06502 }

```

```

06500     namespace Detail {
06501         inline void optimizer_barrier() { keep_memory(); }
06502     } // namespace Detail
06503 #elif defined(_MSC_VER)
06504
06505 #pragma optimize("", off)
06506     template <typename T>
06507     inline void keep_memory(T* p) {
06508         // thanks @milleniumbug
06509         *reinterpret_cast<char volatile*>(p) = *reinterpret_cast<char const volatile*>(p);
06510     }
06511     // TODO equivalent keep_memory()
06512 #pragma optimize("", on)
06513
06514     namespace Detail {
06515         inline void optimizer_barrier() {
06516             std::atomic_thread_fence(std::memory_order_seq_cst);
06517         }
06518     } // namespace Detail
06519
06520 #endif
06521
06522     template <typename T>
06523     inline void deoptimize_value(T&& x) {
06524         keep_memory(&x);
06525     }
06526
06527     template <typename Fn, typename... Args>
06528     inline auto invoke_deoptimized(Fn&& fn, Args&&... args) -> typename
std::enable_if<!std::is_same<void, decltype(fn(args...))>::value::type {
06529         deoptimize_value(std::forward<Fn>(fn) (std::forward<Args...>(args...)));
06530     }
06531
06532     template <typename Fn, typename... Args>
06533     inline auto invoke_deoptimized(Fn&& fn, Args&&... args) -> typename
std::enable_if<std::is_same<void, decltype(fn(args...))>::value::type {
06534         std::forward<Fn>(fn) (std::forward<Args...>(args...));
06535     }
06536 } // namespace Benchmark
06537 } // namespace Catch
06538
06539 // end catch_optimizer.hpp
06540 // start catch_complete_invoke.hpp
06541
06542 // Invoke with a special case for void
06543
06544 #include <type_traits>
06545 #include <utility>
06546
06547 namespace Catch {
06548     namespace Benchmark {
06549         namespace Detail {
06550             template <typename T>
06551             struct CompleteType { using type = T; };
06552             template <>
06553             struct CompleteType<void> { struct type {}; };
06554
06555             template <typename T>
06556             using CompleteType_t = typename CompleteType<T>::type;
06557
06558             template <typename Result>
06559             struct CompleteInvoker {
06560                 template <typename Fun, typename... Args>
06561                 static Result invoke(Fun&& fun, Args&&... args) {
06562                     return std::forward<Fun>(fun) (std::forward<Args>(args)...);
06563                 }
06564             };
06565
06566             template <>
06567             struct CompleteInvoker<void> {
06568                 template <typename Fun, typename... Args>
06569                 static CompleteType_t<void> invoke(Fun&& fun, Args&&... args) {
06570                     std::forward<Fun>(fun) (std::forward<Args>(args)...);
06571                     return {};
06572                 }
06573             };
06574
06575             // invoke and not return void :(
06576             template <typename Fun, typename... Args>
06577             CompleteType_t<FunctionReturnType<Fun, Args...>> complete_invoke(Fun&& fun, Args&&... args)
06578             {
06579                 return CompleteInvoker<FunctionReturnType<Fun,
06580                     Args...>>::invoke(std::forward<Fun>(fun), std::forward<Args>(args)...);
06579             }
06580
06581             const std::string benchmarkErrorMsg = "a benchmark failed to run successfully";
06582         } // namespace Detail

```

```

06583
06584     template <typename Fun>
06585     Detail::CompleteType_t<FunctionReturnType<Fun>> user_code(Fun&& fun) {
06586         CATCH_TRY{
06587             return Detail::complete_invoke(std::forward<Fun>(fun));
06588         } CATCH_CATCH_ALL{
06589             getResultCapture().benchmarkFailed(translateActiveException());
06590             CATCH_RUNTIME_ERROR(Detail::benchmarkErrorMsg);
06591         }
06592     }
06593 } // namespace Benchmark
06594 } // namespace Catch
06595
06596 // end catch_complete_invoke.hpp
06597 namespace Catch {
06598     namespace Benchmark {
06599         namespace Detail {
06600             struct ChronometerConcept {
06601                 virtual void start() = 0;
06602                 virtual void finish() = 0;
06603                 virtual ~ChronometerConcept() = default;
06604             };
06605             template <typename Clock>
06606             struct ChronometerModel final : public ChronometerConcept {
06607                 void start() override { started = Clock::now(); }
06608                 void finish() override { finished = Clock::now(); }
06609
06610                 ClockDuration<Clock> elapsed() const { return finished - started; }
06611
06612                 TimePoint<Clock> started;
06613                 TimePoint<Clock> finished;
06614             };
06615         } // namespace Detail
06616
06617         struct Chronometer {
06618         public:
06619             template <typename Fun>
06620             void measure(Fun&& fun) { measure(std::forward<Fun>(fun), is_callable<Fun(int)>()); }
06621
06622             int runs() const { return k; }
06623
06624             Chronometer(Detail::ChronometerConcept& meter, int k)
06625                 : impl(&meter)
06626                 , k(k) {}
06627
06628         private:
06629             template <typename Fun>
06630             void measure(Fun&& fun, std::false_type) {
06631                 measure([&fun](int) { return fun(); }, std::true_type());
06632             }
06633
06634             template <typename Fun>
06635             void measure(Fun&& fun, std::true_type) {
06636                 Detail::optimizer_barrier();
06637                 impl->start();
06638                 for (int i = 0; i < k; ++i) invoke_deoptimized(fun, i);
06639                 impl->finish();
06640                 Detail::optimizer_barrier();
06641             }
06642
06643             Detail::ChronometerConcept* impl;
06644             int k;
06645         };
06646     } // namespace Benchmark
06647 } // namespace Catch
06648
06649 // end catch_chronometer.hpp
06650 // start catch_environment.hpp
06651
06652 // Environment information
06653
06654 namespace Catch {
06655     namespace Benchmark {
06656         namespace Detail {
06657             template <typename Duration>
06658             struct EnvironmentEstimate {
06659                 Duration mean;
06660                 OutlierClassification outliers;
06661
06662                 template <typename Duration2>
06663                 operator EnvironmentEstimate<Duration2>() const {
06664                     return { mean, outliers };
06665                 }
06666             };
06667             template <typename Clock>
06668             struct Environment {
06669                 using clock_type = Clock;

```

```

06670         EnvironmentEstimate<FloatDuration<Clock>> clock_resolution;
06671         EnvironmentEstimate<FloatDuration<Clock>> clock_cost;
06672     };
06673 } // namespace Benchmark
06674 } // namespace Catch
06675
06676 // end catch_environment.hpp
06677 // start catch_execution_plan.hpp
06678
06679 // Execution plan
06680
06681
06682 // start catch_benchmark_function.hpp
06683
06684 // Dumb std::function implementation for consistent call overhead
06685
06686 #include <cassert>
06687 #include <type_traits>
06688 #include <utility>
06689 #include <memory>
06690
06691 namespace Catch {
06692     namespace Benchmark {
06693         namespace Detail {
06694             template <typename T>
06695             using Decay = typename std::decay<T>::type;
06696             template <typename T, typename U>
06697             struct is_related
06698                 : std::is_same<Decay<T>, Decay<U>> {};
06699
06700             struct BenchmarkFunction {
06701             private:
06702                 struct callable {
06703                     virtual void call(Chronometer meter) const = 0;
06704                     virtual callable* clone() const = 0;
06705                     virtual ~callable() = default;
06706                 };
06707                 template <typename Fun>
06708                 struct model : public callable {
06709                     model(Fun&& fun) : fun(std::move(fun)) {}
06710                     model(Fun const& fun) : fun(fun) {}
06711
06712                     model<Fun>* clone() const override { return new model<Fun>(*this); }
06713
06714                     void call(Chronometer meter) const override {
06715                         call(meter, is_callable<Fun(Chronometer)>());
06716                     }
06717                     void call(Chronometer meter, std::true_type) const {
06718                         fun(meter);
06719                     }
06720                     void call(Chronometer meter, std::false_type) const {
06721                         meter.measure(fun);
06722                     }
06723
06724                     Fun fun;
06725                 };
06726
06727                 struct do_nothing { void operator()() const {} };
06728
06729                 template <typename T>
06730                 BenchmarkFunction(model<T>* c) : f(c) {}
06731
06732             public:
06733                 BenchmarkFunction()
06734                     : f(new model<do_nothing>{ {} }) {}
06735
06736                 template <typename Fun,
06737                     typename std::enable_if<!is_related<Fun, BenchmarkFunction>::value, int>::type =
06738                     0>
06739                     BenchmarkFunction(Fun&& fun)
06740                         : f(new model<typename std::decay<Fun>::type>(std::forward<Fun>(fun))) {}
06741
06742                 BenchmarkFunction(BenchmarkFunction&& that)
06743                     : f(std::move(that.f)) {}
06744
06745                 BenchmarkFunction(BenchmarkFunction const& that)
06746                     : f(that.f->clone()) {}
06747
06748                 BenchmarkFunction& operator=(BenchmarkFunction&& that) {
06749                     f = std::move(that.f);
06750                     return *this;
06751                 }
06752
06753                 BenchmarkFunction& operator=(BenchmarkFunction const& that) {
06754                     f.reset(that.f->clone());
06755                     return *this;
06756                 }
06757             };
06758         }
06759     }
06760 }
06761
06762

```



```

06763         }
06764
06765         void operator()(Chronometer meter) const { f->call(meter); }
06766
06767     private:
06768         std::unique_ptr<callable> f;
06769     };
06770 } // namespace Detail
06771 } // namespace Benchmark
06772 } // namespace Catch
06773
06774 // end catch_benchmark_function.hpp
06775 // start catch_repeat.hpp
06776
06777 // repeat algorithm
06778
06779
06780 #include <type_traits>
06781 #include <utility>
06782
06783 namespace Catch {
06784     namespace Benchmark {
06785         namespace Detail {
06786             template <typename Fun>
06787             struct repeater {
06788                 void operator()(int k) const {
06789                     for (int i = 0; i < k; ++i) {
06790                         fun();
06791                     }
06792                 }
06793                 Fun fun;
06794             };
06795             template <typename Fun>
06796             repeater<typename std::decay<Fun>::type> repeat(Fun&& fun) {
06797                 return { std::forward<Fun>(fun) };
06798             }
06799         } // namespace Detail
06800     } // namespace Benchmark
06801 } // namespace Catch
06802
06803 // end catch_repeat.hpp
06804 // start catch_run_for_at_least.hpp
06805
06806 // Run a function for a minimum amount of time
06807
06808
06809 // start catch_measure.hpp
06810
06811 // Measure
06812
06813
06814 // start catch_timing.hpp
06815
06816 // Timing
06817
06818
06819 #include <tuple>
06820 #include <type_traits>
06821
06822 namespace Catch {
06823     namespace Benchmark {
06824         template <typename Duration, typename Result>
06825         struct Timing {
06826             Duration elapsed;
06827             Result result;
06828             int iterations;
06829         };
06830         template <typename Clock, typename Func, typename... Args>
06831         using TimingOf = Timing<ClockDuration<Clock>, Detail::CompleteType_t<FunctionReturnType<Func,
06832             Args...>>;
06833     } // namespace Benchmark
06834 } // namespace Catch
06835
06836 // end catch_timing.hpp
06837 #include <utility>
06838
06839 namespace Catch {
06840     namespace Benchmark {
06841         namespace Detail {
06842             template <typename Clock, typename Fun, typename... Args>
06843             TimingOf<Clock, Fun, Args...> measure(Fun&& fun, Args&&... args) {
06844                 auto start = Clock::now();
06845                 auto&& r = Detail::complete_invoke(fun, std::forward<Args>(args)...);
06846                 auto end = Clock::now();
06847                 auto delta = end - start;
06848                 return { delta, std::forward<decltype(r)>(r), 1 };
06849             }
06850         }
06851     }
06852 }

```

```

06849         } // namespace Detail
06850     } // namespace Benchmark
06851 } // namespace Catch
06852
06853 // end catch_measure.hpp
06854 #include <utility>
06855 #include <type_traits>
06856
06857 namespace Catch {
06858     namespace Benchmark {
06859         namespace Detail {
06860             template <typename Clock, typename Fun>
06861             TimingOf<Clock, Fun, int> measure_one(Fun&& fun, int iters, std::false_type) {
06862                 return Detail::measure<Clock>(fun, iters);
06863             }
06864             template <typename Clock, typename Fun>
06865             TimingOf<Clock, Fun, Chronometer> measure_one(Fun&& fun, int iters, std::true_type) {
06866                 Detail::ChronometerModel<Clock> meter;
06867                 auto&& result = Detail::complete_invoke(fun, Chronometer(meter, iters));
06868
06869                 return { meter.elapsed(), std::move(result), iters };
06870             }
06871
06872             template <typename Clock, typename Fun>
06873             using run_for_at_least_argument_t = typename
std::conditional<is_callable<Fun(Chronometer)>::value, Chronometer, int>::type;
06874
06875             struct optimized_away_error : std::exception {
06876                 const char* what() const noexcept override {
06877                     return "could not measure benchmark, maybe it was optimized away";
06878                 }
06879             };
06880
06881             template <typename Clock, typename Fun>
06882             TimingOf<Clock, Fun, run_for_at_least_argument_t<Clock, Fun>
run_for_at_least(ClockDuration<Clock> how_long, int seed, Fun&& fun) {
06883                 auto iters = seed;
06884                 while (iters < (1 << 30)) {
06885                     auto&& Timing = measure_one<Clock>(fun, iters, is_callable<Fun(Chronometer)>());
06886
06887                     if (Timing.elapsed >= how_long) {
06888                         return { Timing.elapsed, std::move(Timing.result), iters };
06889                     }
06890                     iters *= 2;
06891                 }
06892                 Catch::throw_exception(optimized_away_error{});
06893             }
06894         } // namespace Detail
06895     } // namespace Benchmark
06896 } // namespace Catch
06897
06898 // end catch_run_for_at_least.hpp
06899 #include <algorithm>
06900 #include <iterator>
06901
06902 namespace Catch {
06903     namespace Benchmark {
06904         template <typename Duration>
06905         struct ExecutionPlan {
06906             int iterations_per_sample;
06907             Duration estimated_duration;
06908             Detail::BenchmarkFunction benchmark;
06909             Duration warmup_time;
06910             int warmup_iterations;
06911
06912             template <typename Duration2>
06913             operator ExecutionPlan<Duration2>() const {
06914                 return { iterations_per_sample, estimated_duration, benchmark, warmup_time,
warmup_iterations };
06915             }
06916
06917             template <typename Clock>
06918             std::vector<FloatDuration<Clock>> run(const IConfig &cfg, Environment<FloatDuration<Clock>>
env) const {
06919                 // warmup a bit
06920
06921                 Detail::run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(warmup_time),
warmup_iterations, Detail::repeat(now<Clock>{}));
06922
06923                 std::vector<FloatDuration<Clock>> times;
06924                 times.reserve(cfg.benchmarkSamples());
06925                 std::generate_n(std::back_inserter(times), cfg.benchmarkSamples(), [this, env] {
06926                     Detail::ChronometerModel<Clock> model;
06927                     this->benchmark(Chronometer(model, iterations_per_sample));
06928                     auto sample_time = model.elapsed() - env.clock_cost.mean;
06929                     if (sample_time < FloatDuration<Clock>::zero()) sample_time =
FloatDuration<Clock>::zero();

```

```

06929         return sample_time / iterations_per_sample;
06930     });
06931     return times;
06932 }
06933 };
06934 } // namespace Benchmark
06935 } // namespace Catch
06936
06937 // end catch_execution_plan.hpp
06938 // start catch_estimate_clock.hpp
06939
06940 // Environment measurement
06941
06942
06943 // start catch_stats.hpp
06944
06945 // Statistical analysis tools
06946
06947
06948 #include <algorithm>
06949 #include <functional>
06950 #include <vector>
06951 #include <iterator>
06952 #include <numeric>
06953 #include <tuple>
06954 #include <cmath>
06955 #include <utility>
06956 #include <cstdint>
06957 #include <random>
06958
06959 namespace Catch {
06960     namespace Benchmark {
06961         namespace Detail {
06962             using sample = std::vector<double>;
06963
06964             double weighted_average_quantile(int k, int q, std::vector<double>::iterator first,
06965 std::vector<double>::iterator last);
06966
06967             template <typename Iterator>
06968             OutlierClassification classify_outliers(Iterator first, Iterator last) {
06969                 std::vector<double> copy(first, last);
06970
06971                 auto q1 = weighted_average_quantile(1, 4, copy.begin(), copy.end());
06972                 auto q3 = weighted_average_quantile(3, 4, copy.begin(), copy.end());
06973                 auto iqr = q3 - q1;
06974                 auto los = q1 - (iqr * 3.);
06975                 auto lom = q1 - (iqr * 1.5);
06976                 auto him = q3 + (iqr * 1.5);
06977                 auto his = q3 + (iqr * 3.);
06978
06979                 OutlierClassification o;
06980                 for (; first != last; ++first) {
06981                     auto& t = *first;
06982                     if (t < los) ++o.low_severe;
06983                     else if (t < lom) ++o.low_mild;
06984                     else if (t > his) ++o.high_severe;
06985                     else if (t > him) ++o.high_mild;
06986                     ++o.samples_seen;
06987                 }
06988                 return o;
06989             }
06990
06991             template <typename Iterator>
06992             double mean(Iterator first, Iterator last) {
06993                 auto count = last - first;
06994                 double sum = std::accumulate(first, last, 0.);
06995                 return sum / count;
06996             }
06997
06998             template <typename URng, typename Iterator, typename Estimator>
06999             sample resample(URng& rng, int resamples, Iterator first, Iterator last, Estimator&
07000 estimator) {
07001                 auto n = last - first;
07002                 std::uniform_int_distribution<decltype(n)> dist(0, n - 1);
07003
07004                 sample out;
07005                 out.reserve(resamples);
07006                 std::generate_n(std::back_inserter(out), resamples, [n, first, &estimator, &dist,
07007 &rng] {
07008                     std::vector<double> resampled;
07009                     resampled.reserve(n);
07010                     std::generate_n(std::back_inserter(resampled), n, [first, &dist, &rng] { return
07011 first[dist(rng)]; });
07012                     return estimator(resampled.begin(), resampled.end());
07013                 });
07014                 std::sort(out.begin(), out.end());
07015                 return out;
07016             }
07017         }
07018     }
07019 }

```

```

07012     }
07013
07014     template <typename Estimator, typename Iterator>
07015     sample jackknife(Estimator&& estimator, Iterator first, Iterator last) {
07016         auto n = last - first;
07017         auto second = std::next(first);
07018         sample results;
07019         results.reserve(n);
07020
07021         for (auto it = first; it != last; ++it) {
07022             std::iter_swap(it, first);
07023             results.push_back(estimator(second, last));
07024         }
07025
07026         return results;
07027     }
07028
07029     inline double normal_cdf(double x) {
07030         return std::erfc(-x / std::sqrt(2.0)) / 2.0;
07031     }
07032
07033     double erfc_inv(double x);
07034
07035     double normal_quantile(double p);
07036
07037     template <typename Iterator, typename Estimator>
07038     Estimate<double> bootstrap(double confidence_level, Iterator first, Iterator last, sample
const& resample, Estimator&& estimator) {
07039         auto n_samples = last - first;
07040
07041         double point = estimator(first, last);
07042         // Degenerate case with a single sample
07043         if (n_samples == 1) return { point, point, point, confidence_level };
07044
07045         sample jack = jackknife(estimator, first, last);
07046         double jack_mean = mean(jack.begin(), jack.end());
07047         double sum_squares, sum_cubes;
07048         std::tie(sum_squares, sum_cubes) = std::accumulate(jack.begin(), jack.end(),
std::make_pair(0., 0.), [jack_mean](std::pair<double, double> sqcb, double x) -> std::pair<double,
double> {
07049             auto d = jack_mean - x;
07050             auto d2 = d * d;
07051             auto d3 = d2 * d;
07052             return { sqcb.first + d2, sqcb.second + d3 };
07053         });
07054
07055         double accel = sum_cubes / (6 * std::pow(sum_squares, 1.5));
07056         int n = static_cast<int>(resample.size());
07057         double prob_n = std::count_if(resample.begin(), resample.end(), [point](double x) {
return x < point; }) / (double)n;
07058         // degenerate case with uniform samples
07059         if (prob_n == 0) return { point, point, point, confidence_level };
07060
07061         double bias = normal_quantile(prob_n);
07062         double z1 = normal_quantile((1. - confidence_level) / 2.);
07063
07064         auto cumn = [n](double x) -> int {
07065             return std::lround(normal_cdf(x) * n); };
07066         auto a = [bias, accel](double b) { return bias + b / (1. - accel * b); };
07067         double b1 = bias + z1;
07068         double b2 = bias - z1;
07069         double a1 = a(b1);
07070         double a2 = a(b2);
07071         auto lo = (std::max)(cumn(a1), 0);
07072         auto hi = (std::min)(cumn(a2), n - 1);
07073
07074         return { point, resample[lo], resample[hi], confidence_level };
07075     }
07076
07077     double outlier_variance(Estimate<double> mean, Estimate<double> stddev, int n);
07078
07079     struct bootstrap_analysis {
07080         Estimate<double> mean;
07081         Estimate<double> standard_deviation;
07082         double outlier_variance;
07083     };
07084
07085     bootstrap_analysis analyse_samples(double confidence_level, int n_resamples,
std::vector<double>::iterator first, std::vector<double>::iterator last);
07086     } // namespace Detail
07087     } // namespace Benchmark
07088 } // namespace Catch
07089
07090 // end catch_stats.hpp
07091 #include <algorithm>
07092 #include <iterator>
07093 #include <tuple>

```

```

07094 #include <vector>
07095 #include <cmath>
07096
07097 namespace Catch {
07098     namespace Benchmark {
07099         namespace Detail {
07100             template <typename Clock>
07101             std::vector<double> resolution(int k) {
07102                 std::vector<TimePoint<Clock>> times;
07103                 times.reserve(k + 1);
07104                 std::generate_n(std::back_inserter(times), k + 1, now<Clock>{});
07105
07106                 std::vector<double> deltas;
07107                 deltas.reserve(k);
07108                 std::transform(std::next(times.begin()), times.end(), times.begin(),
07109                     std::back_inserter(deltas),
07110                     [](TimePoint<Clock> a, TimePoint<Clock> b) { return static_cast<double>((a -
07111 b).count()); });
07112
07113                 return deltas;
07114             }
07115
07116             const auto warmup_iterations = 10000;
07117             const auto warmup_time = std::chrono::milliseconds(100);
07118             const auto minimum_ticks = 1000;
07119             const auto warmup_seed = 10000;
07120             const auto clock_resolution_estimation_time = std::chrono::milliseconds(500);
07121             const auto clock_cost_estimation_time_limit = std::chrono::seconds(1);
07122             const auto clock_cost_estimation_tick_limit = 100000;
07123             const auto clock_cost_estimation_time = std::chrono::milliseconds(10);
07124             const auto clock_cost_estimation_iterations = 10000;
07125
07126             template <typename Clock>
07127             int warmup() {
07128                 run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(warmup_time), warmup_seed,
07129                     &resolution<Clock>)
07130                     .iterations;
07131             }
07132
07133             template <typename Clock>
07134             EnvironmentEstimate<FloatDuration<Clock>> estimate_clock_resolution(int iterations) {
07135                 auto r =
07136                     run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(clock_resolution_estimation_time),
07137                         iterations, &resolution<Clock>)
07138                         .result;
07139                 return {
07140                     FloatDuration<Clock>(mean(r.begin(), r.end())),
07141                     classify_outliers(r.begin(), r.end()),
07142                 };
07143             }
07144
07145             template <typename Clock>
07146             EnvironmentEstimate<FloatDuration<Clock>> estimate_clock_cost(FloatDuration<Clock>
07147 resolution) {
07148                 auto time_limit = (std::min)(
07149                     resolution * clock_cost_estimation_tick_limit,
07150                     FloatDuration<Clock>(clock_cost_estimation_time_limit));
07151                 auto time_clock = [](int k) {
07152                     return Detail::measure<Clock>([k] {
07153                         for (int i = 0; i < k; ++i) {
07154                             volatile auto ignored = Clock::now();
07155                             (void)ignored;
07156                         }
07157                     }).elapsed;
07158                 };
07159                 time_clock(1);
07160                 int iters = clock_cost_estimation_iterations;
07161                 auto&& r =
07162                     run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(clock_cost_estimation_time),
07163                         iters, time_clock);
07164                 std::vector<double> times;
07165                 int nsamples = static_cast<int>(std::ceil(time_limit / r.elapsed));
07166                 times.reserve(nsamples);
07167                 std::generate_n(std::back_inserter(times), nsamples, [time_clock, &r] {
07168                     return static_cast<double>((time_clock(r.iterations) / r.iterations).count());
07169                 });
07170                 return {
07171                     FloatDuration<Clock>(mean(times.begin(), times.end())),
07172                     classify_outliers(times.begin(), times.end()),
07173                 };
07174             }
07175
07176             template <typename Clock>
07177             EnvironmentEstimate<FloatDuration<Clock>> measure_environment() {
07178                 static EnvironmentEstimate<FloatDuration<Clock>>* env = nullptr;
07179                 if (env) {
07180                     return *env;
07181                 }
07182             }

```

```

07173
07174         auto iters = Detail::warmup<Clock>();
07175         auto resolution = Detail::estimate_clock_resolution<Clock>(iters);
07176         auto cost = Detail::estimate_clock_cost<Clock>(resolution.mean);
07177
07178         env = new Environment<FloatDuration<Clock>>{ resolution, cost };
07179         return *env;
07180     }
07181     } // namespace Detail
07182 } // namespace Benchmark
07183 } // namespace Catch
07184
07185 // end catch_estimate_clock.hpp
07186 // start catch_analyse.hpp
07187
07188 // Run and analyse one benchmark
07189
07190
07191 // start catch_sample_analysis.hpp
07192
07193 // Benchmark results
07194
07195
07196 #include <algorithm>
07197 #include <vector>
07198 #include <string>
07199 #include <iterator>
07200
07201 namespace Catch {
07202     namespace Benchmark {
07203         template <typename Duration>
07204         struct SampleAnalysis {
07205             std::vector<Duration> samples;
07206             Estimate<Duration> mean;
07207             Estimate<Duration> standard_deviation;
07208             OutlierClassification outliers;
07209             double outlier_variance;
07210
07211             template <typename Duration2>
07212             operator SampleAnalysis<Duration2>() const {
07213                 std::vector<Duration2> samples2;
07214                 samples2.reserve(samples.size());
07215                 std::transform(samples.begin(), samples.end(), std::back_inserter(samples2),
07216                     [](Duration d) { return Duration2(d); });
07217                 return {
07218                     std::move(samples2),
07219                     mean,
07220                     standard_deviation,
07221                     outliers,
07222                     outlier_variance,
07223                 };
07224             }
07225         } // namespace Benchmark
07226     } // namespace Catch
07227
07228 // end catch_sample_analysis.hpp
07229 #include <algorithm>
07230 #include <iterator>
07231 #include <vector>
07232
07233 namespace Catch {
07234     namespace Benchmark {
07235         namespace Detail {
07236             template <typename Duration, typename Iterator>
07237             SampleAnalysis<Duration> analyse(const IConfig &cfg, Environment<Duration>, Iterator
07238 first, Iterator last) {
07239                 if (!cfg.benchmarkNoAnalysis()) {
07240                     std::vector<double> samples;
07241                     samples.reserve(last - first);
07242                     std::transform(first, last, std::back_inserter(samples), [](Duration d) { return
07243 d.count(); });
07244
07245                     auto analysis =
07246 Catch::Benchmark::Detail::analyse_samples(cfg.benchmarkConfidenceInterval(), cfg.benchmarkResamples(),
07247 samples.begin(), samples.end());
07248
07249                     auto outliers = Catch::Benchmark::Detail::classify_outliers(samples.begin(),
07250 samples.end());
07251
07252                     auto wrap_estimate = [](Estimate<double> e) {
07253                         return Estimate<Duration> {
07254                             Duration(e.point),
07255                             Duration(e.lower_bound),
07256                             Duration(e.upper_bound),
07257                             e.confidence_interval,
07258                         };
07259                     };
07260                 }
07261             }
07262         }
07263     }
07264 }

```

```

07254         std::vector<Duration> samples2;
07255         samples2.reserve(samples.size());
07256         std::transform(samples.begin(), samples.end(), std::back_inserter(samples2),
[] (double d) { return Duration(d); });
07257         return {
07258             std::move(samples2),
07259             wrap_estimate(analysis.mean),
07260             wrap_estimate(analysis.standard_deviation),
07261             outliers,
07262             analysis.outlier_variance,
07263         };
07264     } else {
07265         std::vector<Duration> samples;
07266         samples.reserve(last - first);
07267
07268         Duration mean = Duration(0);
07269         int i = 0;
07270         for (auto it = first; it < last; ++it, ++i) {
07271             samples.push_back(Duration(*it));
07272             mean += Duration(*it);
07273         }
07274         mean /= i;
07275
07276         return {
07277             std::move(samples),
07278             Estimate<Duration>{mean, mean, mean, 0.0},
07279             Estimate<Duration>{Duration(0), Duration(0), Duration(0), 0.0},
07280             OutlierClassification{},
07281             0.0
07282         };
07283     }
07284 }
07285 } // namespace Detail
07286 } // namespace Benchmark
07287 } // namespace Catch
07288
07289 // end catch_analyse.hpp
07290 #include <algorithm>
07291 #include <functional>
07292 #include <string>
07293 #include <vector>
07294 #include <cmath>
07295
07296 namespace Catch {
07297     namespace Benchmark {
07298         struct Benchmark {
07299             Benchmark(std::string &&name)
07300                 : name(std::move(name)) {}
07301
07302             template <class FUN>
07303             Benchmark(std::string &&name, FUN &&func)
07304                 : fun(std::move(func)), name(std::move(name)) {}
07305
07306             template <typename Clock>
07307             ExecutionPlan<FloatDuration<Clock>> prepare(const IConfig &cfg,
Environment<FloatDuration<Clock>> env) const {
07308                 auto min_time = env.clock_resolution.mean * Detail::minimum_ticks;
07309                 auto run_time = std::max(min_time,
std::chrono::duration_cast<decltype(min_time)>(cfg.benchmarkWarmupTime()));
07310                 auto&& test =
Detail::run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(run_time), 1, fun);
07311                 int new_iters = static_cast<int>(std::ceil(min_time * test.iterations /
test.elapsed));
07312                 return { new_iters, test.elapsed / test.iterations * new_iters *
cfg.benchmarkSamples(), fun,
std::chrono::duration_cast<FloatDuration<Clock>>(cfg.benchmarkWarmupTime()), Detail::warmup_iterations
};
07313     }
07314
07315     template <typename Clock = default_clock>
07316     void run() {
07317         IConfigPtr cfg = getCurrentContext().getConfig();
07318
07319         auto env = Detail::measure_environment<Clock>();
07320
07321         getResultCapture().benchmarkPreparing(name);
07322         CATCH_TRY{
07323             auto plan = user_code([&] {
07324                 return prepare<Clock>(*cfg, env);
07325             });
07326
07327             BenchmarkInfo info {
07328                 name,
07329                 plan.estimated_duration.count(),
07330                 plan.iterations_per_sample,
07331                 cfg->benchmarkSamples(),
07332                 cfg->benchmarkResamples(),

```

```

07333         env.clock_resolution.mean.count(),
07334         env.clock_cost.mean.count()
07335     };
07336
07337     getResultCapture().benchmarkStarting(info);
07338
07339     auto samples = user_code([&] {
07340         return plan.template run<Clock>(*cfg, env);
07341     });
07342
07343     auto analysis = Detail::analyse(*cfg, env, samples.begin(), samples.end());
07344     BenchmarkStats<FloatDuration<Clock>> stats{ info, analysis.samples, analysis.mean,
analysis.standard_deviation, analysis.outliers, analysis.outlier_variance };
07345     getResultCapture().benchmarkEnded(stats);
07346
07347     } CATCH_CATCH_ALL{
07348         if (translateActiveException() != Detail::benchmarkErrorMsg) // benchmark errors
have been reported, otherwise rethrow.
07349         std::rethrow_exception(std::current_exception());
07350     }
07351 }
07352
07353 // sets lambda to be used in fun *and* executes benchmark!
07354 template <typename Fun,
07355     typename std::enable_if<!Detail::is_related<Fun, Benchmark>::value, int>::type = 0>
07356     Benchmark & operator=(Fun func) {
07357     fun = Detail::BenchmarkFunction(func);
07358     run();
07359     return *this;
07360 }
07361
07362 explicit operator bool() {
07363     return true;
07364 }
07365
07366 private:
07367     Detail::BenchmarkFunction fun;
07368     std::string name;
07369 };
07370 }
07371 } // namespace Catch
07372
07373 #define INTERNAL_CATCH_GET_1_ARG(arg1, arg2, ...) arg1
07374 #define INTERNAL_CATCH_GET_2_ARG(arg1, arg2, ...) arg2
07375
07376 #define INTERNAL_CATCH_BENCHMARK(BenchmarkName, name, benchmarkIndex)\
07377     if( Catch::Benchmark::Benchmark BenchmarkName{name} ) {\
07378         BenchmarkName = [&](int benchmarkIndex)
07379
07380 #define INTERNAL_CATCH_BENCHMARK_ADVANCED(BenchmarkName, name)\
07381     if( Catch::Benchmark::Benchmark BenchmarkName{name} ) {\
07382         BenchmarkName = [&]
07383
07384 // end catch_benchmark.hpp
07385 // start catch_constructor.hpp
07386
07387 // Constructor and destructor helpers
07388
07389
07390 #include <type_traits>
07391
07392 namespace Catch {
07393     namespace Benchmark {
07394         namespace Detail {
07395             template <typename T, bool Destruct>
07396             struct ObjectStorage
07397             {
07398                 ObjectStorage() : data() {}
07399
07400                 ObjectStorage(const ObjectStorage& other)
07401                 {
07402                     new(&data) T(other.stored_object());
07403                 }
07404
07405                 ObjectStorage(ObjectStorage&& other)
07406                 {
07407                     new(&data) T(std::move(other.stored_object()));
07408                 }
07409
07410                 ~ObjectStorage() { destruct_on_exit<T>(); }
07411
07412                 template <typename... Args>
07413                 void construct(Args&&... args)
07414                 {
07415                     new (&data) T(std::forward<Args>(args)...);
07416                 }
07417

```



```

07418         template <bool AllowManualDestruction = !Destruct>
07419         typename std::enable_if<AllowManualDestruction>::type destruct()
07420         {
07421             stored_object().~T();
07422         }
07423
07424     private:
07425         // If this is a constructor benchmark, destruct the underlying object
07426         template <typename U>
07427         void destruct_on_exit(typename std::enable_if<Destruct, U>::type* = 0) {
07428             destruct<true>(); }
07429         // Otherwise, don't
07430         template <typename U>
07431         void destruct_on_exit(typename std::enable_if<!Destruct, U>::type* = 0) { }
07432
07433         T& stored_object() {
07434             return *static_cast<T*>(static_cast<void*>(&data));
07435         }
07436
07437         T const& stored_object() const {
07438             return *static_cast<T*>(static_cast<void*>(&data));
07439         }
07440
07441         struct { alignas(T) unsigned char data[sizeof(T)]; } data;
07442     };
07443
07444     template <typename T>
07445     using storage_for = Detail::ObjectStorage<T, true>;
07446
07447     template <typename T>
07448     using destructable_object = Detail::ObjectStorage<T, false>;
07449 }
07450
07451 // end catch_constructor.hpp
07452 // end catch_benchmarking_all.hpp
07453 #endif
07454
07455 #endif // ! CATCH_CONFIG_IMPL_ONLY
07456
07457 #ifdef CATCH_IMPL
07458 // start catch_impl.hpp
07459
07460 #ifdef __clang__
07461 #pragma clang diagnostic push
07462 #pragma clang diagnostic ignored "-Wweak-vtables"
07463 #endif
07464
07465 // Keep these here for external reporters
07466 // start catch_test_case_tracker.h
07467
07468 #include <string>
07469 #include <vector>
07470 #include <memory>
07471
07472 namespace Catch {
07473     namespace TestCaseTracking {
07474         struct NameAndLocation {
07475             std::string name;
07476             SourceLineInfo location;
07477
07478             NameAndLocation( std::string const& _name, SourceLineInfo const& _location );
07479             friend bool operator==(NameAndLocation const& lhs, NameAndLocation const& rhs) {
07480                 return lhs.name == rhs.name
07481                     && lhs.location == rhs.location;
07482             }
07483         };
07484     };
07485
07486     class ITracker;
07487
07488     using ITrackerPtr = std::shared_ptr<ITracker>;
07489
07490     class ITracker {
07491     public:
07492         NameAndLocation m_nameAndLocation;
07493
07494         ITracker(NameAndLocation const& nameAndLoc) :
07495             m_nameAndLocation(nameAndLoc)
07496         {}
07497
07498         // static queries
07499         NameAndLocation const& nameAndLocation() const {
07500             return m_nameAndLocation;
07501         }
07502     };
07503

```

```

07504     virtual ~ITracker();
07505
07506     // dynamic queries
07507     virtual bool isComplete() const = 0; // Successfully completed or failed
07508     virtual bool isSuccessfullyCompleted() const = 0;
07509     virtual bool isOpen() const = 0; // Started but not complete
07510     virtual bool hasChildren() const = 0;
07511     virtual bool hasStarted() const = 0;
07512
07513     virtual ITracker& parent() = 0;
07514
07515     // actions
07516     virtual void close() = 0; // Successfully complete
07517     virtual void fail() = 0;
07518     virtual void markAsNeedingAnotherRun() = 0;
07519
07520     virtual void addChild( ITrackerPtr const& child ) = 0;
07521     virtual ITrackerPtr findChild( NameAndLocation const& nameAndLocation ) = 0;
07522     virtual void openChild() = 0;
07523
07524     // Debug/ checking
07525     virtual bool isSectionTracker() const = 0;
07526     virtual bool isGeneratorTracker() const = 0;
07527 };
07528
07529 class TrackerContext {
07530
07531     enum RunState {
07532         NotStarted,
07533         Executing,
07534         CompletedCycle
07535     };
07536
07537     ITrackerPtr m_rootTracker;
07538     ITracker* m_currentTracker = nullptr;
07539     RunState m_runState = NotStarted;
07540
07541 public:
07542
07543     ITracker& startRun();
07544     void endRun();
07545
07546     void startCycle();
07547     void completeCycle();
07548
07549     bool completedCycle() const;
07550     ITracker& currentTracker();
07551     void setCurrentTracker( ITracker* tracker );
07552 };
07553
07554 class TrackerBase : public ITracker {
07555 protected:
07556     enum CycleState {
07557         NotStarted,
07558         Executing,
07559         ExecutingChildren,
07560         NeedsAnotherRun,
07561         CompletedSuccessfully,
07562         Failed
07563     };
07564
07565     using Children = std::vector<ITrackerPtr>;
07566     TrackerContext& m_ctx;
07567     ITracker* m_parent;
07568     Children m_children;
07569     CycleState m_runState = NotStarted;
07570
07571 public:
07572     TrackerBase( NameAndLocation const& nameAndLocation, TrackerContext& ctx, ITracker* parent );
07573
07574     bool isComplete() const override;
07575     bool isSuccessfullyCompleted() const override;
07576     bool isOpen() const override;
07577     bool hasChildren() const override;
07578     bool hasStarted() const override {
07579         return m_runState != NotStarted;
07580     }
07581
07582     void addChild( ITrackerPtr const& child ) override;
07583
07584     ITrackerPtr findChild( NameAndLocation const& nameAndLocation ) override;
07585     ITracker& parent() override;
07586
07587     void openChild() override;
07588
07589     bool isSectionTracker() const override;
07590     bool isGeneratorTracker() const override;

```

```

07591
07592     void open();
07593
07594     void close() override;
07595     void fail() override;
07596     void markAsNeedingAnotherRun() override;
07597
07598     private:
07599         void moveToParent();
07600         void moveToThis();
07601     };
07602
07603     class SectionTracker : public TrackerBase {
07604         std::vector<std::string> m_filters;
07605         std::string m_trimmed_name;
07606     public:
07607         SectionTracker( NameAndLocation const& nameAndLocation, TrackerContext& ctx, ITracker* parent
07608     );
07609
07610         bool isSectionTracker() const override;
07611
07612         bool isComplete() const override;
07613
07614         static SectionTracker& acquire( TrackerContext& ctx, NameAndLocation const& nameAndLocation );
07615
07616         void tryOpen();
07617
07618         void addInitialFilters( std::vector<std::string> const& filters );
07619         void addNextFilters( std::vector<std::string> const& filters );
07620         std::vector<std::string> const& getFilters() const;
07621         std::string const& trimmedName() const;
07622     };
07623
07624
07625 } // namespace TestCaseTracking
07626
07627 using TestCaseTracking::ITracker;
07628 using TestCaseTracking::TrackerContext;
07629 using TestCaseTracking::SectionTracker;
07630
07631 } // namespace Catch
07632
07633 // end catch_test_case_tracker.h
07634
07635 // start catch_leak_detector.h
07636
07637 namespace Catch {
07638
07639     struct LeakDetector {
07640         LeakDetector();
07641         ~LeakDetector();
07642     };
07643
07644 }
07645 // end catch_leak_detector.h
07646 // Cpp files will be included in the single-header file here
07647 // start catch_stats.cpp
07648
07649 // Statistical analysis tools
07650
07651 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
07652
07653 #include <cassert>
07654 #include <random>
07655
07656 #if defined(CATCH_CONFIG_USE_ASYNC)
07657 #include <future>
07658 #endif
07659
07660 namespace {
07661     double erf_inv(double x) {
07662         // Code accompanying the article "Approximating the erfinv function" in GPU Computing Gems,
07663         Volume 2
07664         double w, p;
07665
07666         w = -log((1.0 - x) * (1.0 + x));
07667
07668         if (w < 6.250000) {
07669             w = w - 3.125000;
07670             p = -3.6444120640178196996e-21;
07671             p = -1.685059138182016589e-19 + p * w;
07672             p = 1.2858480715256400167e-18 + p * w;
07673             p = 1.115787767802518096e-17 + p * w;
07674             p = -1.333171662854620906e-16 + p * w;
07675             p = 2.0972767875968561637e-17 + p * w;
07676             p = 6.6376381343583238325e-15 + p * w;
07677             p = -4.0545662729752068639e-14 + p * w;
07678             p = -8.1519341976054721522e-14 + p * w;

```

```

07678         p = 2.6335093153082322977e-12 + p * w;
07679         p = -1.2975133253453532498e-11 + p * w;
07680         p = -5.4154120542946279317e-11 + p * w;
07681         p = 1.051212273321532285e-09 + p * w;
07682         p = -4.1126339803469836976e-09 + p * w;
07683         p = -2.9070369957882005086e-08 + p * w;
07684         p = 4.2347877827932403518e-07 + p * w;
07685         p = -1.3654692000834678645e-06 + p * w;
07686         p = -1.3882523362786468719e-05 + p * w;
07687         p = 0.0001867342080340571352 + p * w;
07688         p = -0.00074070253416626697512 + p * w;
07689         p = -0.0060336708714301490533 + p * w;
07690         p = 0.24015818242558961693 + p * w;
07691         p = 1.6536545626831027356 + p * w;
07692     } else if (w < 16.000000) {
07693         w = sqrt(w) - 3.250000;
07694         p = 2.2137376921775787049e-09;
07695         p = 9.0756561938885390979e-08 + p * w;
07696         p = -2.7517406297064545428e-07 + p * w;
07697         p = 1.8239629214389227755e-08 + p * w;
07698         p = 1.5027403968909827627e-06 + p * w;
07699         p = -4.013867526981545969e-06 + p * w;
07700         p = 2.9234449089955446044e-06 + p * w;
07701         p = 1.2475304481671778723e-05 + p * w;
07702         p = -4.7318229009055733981e-05 + p * w;
07703         p = 6.8284851459573175448e-05 + p * w;
07704         p = 2.4031110387097893999e-05 + p * w;
07705         p = -0.0003550375203628474796 + p * w;
07706         p = 0.00095328937973738049703 + p * w;
07707         p = -0.0016882755560235047313 + p * w;
07708         p = 0.0024914420961078508066 + p * w;
07709         p = -0.0037512085075692412107 + p * w;
07710         p = 0.005370914553590063617 + p * w;
07711         p = 1.0052589676941592334 + p * w;
07712         p = 3.0838856104922207635 + p * w;
07713     } else {
07714         w = sqrt(w) - 5.000000;
07715         p = -2.7109920616438573243e-11;
07716         p = -2.5556418169965252055e-10 + p * w;
07717         p = 1.5076572693500548083e-09 + p * w;
07718         p = -3.7894654401267369937e-09 + p * w;
07719         p = 7.6157012080783393804e-09 + p * w;
07720         p = -1.4960026627149240478e-08 + p * w;
07721         p = 2.9147953450901080826e-08 + p * w;
07722         p = -6.7711997758452339498e-08 + p * w;
07723         p = 2.2900482228026654717e-07 + p * w;
07724         p = -9.9298272942317002539e-07 + p * w;
07725         p = 4.5260625972231537039e-06 + p * w;
07726         p = -1.9681778105531670567e-05 + p * w;
07727         p = 7.5995277030017761139e-05 + p * w;
07728         p = -0.00021503011930044477347 + p * w;
07729         p = -0.00013871931833623122026 + p * w;
07730         p = 1.0103004648645343977 + p * w;
07731         p = 4.8499064014085844221 + p * w;
07732     }
07733     return p * x;
07734 }
07735
07736 double standard_deviation(std::vector<double>::iterator first, std::vector<double>::iterator last)
07737 {
07738     auto m = Catch::Benchmark::Detail::mean(first, last);
07739     double variance = std::accumulate(first, last, 0., [m](double a, double b) {
07740         double diff = b - m;
07741         return a + diff * diff;
07742     }) / (last - first);
07743     return std::sqrt(variance);
07744 }
07745
07746 namespace Catch {
07747     namespace Benchmark {
07748         namespace Detail {
07751             double weighted_average_quantile(int k, int q, std::vector<double>::iterator first,
07752 std::vector<double>::iterator last) {
07753                 auto count = last - first;
07754                 double idx = (count - 1) * k / static_cast<double>(q);
07755                 int j = static_cast<int>(idx);
07756                 double g = idx - j;
07757                 std::nth_element(first, first + j, last);
07758                 auto xj = first[j];
07759                 if (g == 0) return xj;
07760                 auto xj1 = *std::min_element(first + (j + 1), last);
07761                 return xj + g * (xj1 - xj);
07762             }

```

```

07763
07764     double erfc_inv(double x) {
07765         return erf_inv(1.0 - x);
07766     }
07767
07768     double normal_quantile(double p) {
07769         static const double ROOT_TWO = std::sqrt(2.0);
07770
07771         double result = 0.0;
07772         assert(p >= 0 && p <= 1);
07773         if (p < 0 || p > 1) {
07774             return result;
07775         }
07776
07777         result = -erfc_inv(2.0 * p);
07778         // result *= normal distribution standard deviation (1.0) * sqrt(2)
07779         result *= /*sd * */ ROOT_TWO;
07780         // result += normal distribution mean (0)
07781         return result;
07782     }
07783
07784     double outlier_variance(Estimate<double> mean, Estimate<double> stddev, int n) {
07785         double sb = stddev.point();
07786         double mn = mean.point() / n;
07787         double mg_min = mn / 2.;
07788         double sg = (std::min)(mg_min / 4., sb / std::sqrt(n));
07789         double sg2 = sg * sg;
07790         double sb2 = sb * sb;
07791
07792         auto c_max = [n, mn, sb2, sg2](double x) -> double {
07793             double k = mn - x;
07794             double d = k * k;
07795             double nd = n * d;
07796             double k0 = -n * nd;
07797             double k1 = sb2 - n * sg2 + nd;
07798             double det = k1 * k1 - 4 * sg2 * k0;
07799             return (int)(-2. * k0 / (k1 + std::sqrt(det)));
07800         };
07801
07802         auto var_out = [n, sb2, sg2](double c) {
07803             double nc = n - c;
07804             return (nc / n) * (sb2 - nc * sg2);
07805         };
07806
07807         return (std::min)(var_out(1), var_out((std::min)(c_max(0.), c_max(mg_min)))) / sb2;
07808     }
07809
07810     bootstrap_analysis analyse_samples(double confidence_level, int n_resamples,
std::vector<double>::iterator first, std::vector<double>::iterator last) {
07811         CATCH_INTERNAL_START_WARNINGS_SUPPRESSION
07812         CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS
07813         static std::random_device entropy;
07814         CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
07815
07816         auto n = static_cast<int>(last - first); // seriously, one can't use integral types
without hell in C++
07817
07818         auto mean = &Detail::mean<std::vector<double>::iterator>;
07819         auto stddev = &standard_deviation;
07820
07821         #if defined(CATCH_CONFIG_USE_ASYNC)
07822         auto Estimate = [=](double(*f)(std::vector<double>::iterator,
std::vector<double>::iterator)) {
07823             auto seed = entropy();
07824             return std::async(std::launch::async, [=] {
07825                 std::mt19937 rng(seed);
07826                 auto resampled = resample(rng, n_resamples, first, last, f);
07827                 return bootstrap(confidence_level, first, last, resampled, f);
07828             });
07829         };
07830
07831         auto mean_future = Estimate(mean);
07832         auto stddev_future = Estimate(stddev);
07833
07834         auto mean_estimate = mean_future.get();
07835         auto stddev_estimate = stddev_future.get();
07836     #else
07837         auto Estimate = [=](double(*f)(std::vector<double>::iterator,
std::vector<double>::iterator)) {
07838             auto seed = entropy();
07839             std::mt19937 rng(seed);
07840             auto resampled = resample(rng, n_resamples, first, last, f);
07841             return bootstrap(confidence_level, first, last, resampled, f);
07842         };
07843
07844         auto mean_estimate = Estimate(mean);
07845         auto stddev_estimate = Estimate(stddev);

```

```

07846 #endif // CATCH_USE_ASYNC
07847
07848         double outlier_variance = Detail::outlier_variance(mean_estimate, stddev_estimate, n);
07849
07850         return { mean_estimate, stddev_estimate, outlier_variance };
07851     }
07852 } // namespace Detail
07853 } // namespace Benchmark
07854 } // namespace Catch
07855
07856 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
07857 // end catch_stats.cpp
07858 // start catch_approx.cpp
07859
07860 #include <cmath>
07861 #include <limits>
07862
07863 namespace {
07864
07865     // Performs equivalent check of std::fabs(lhs - rhs) <= margin
07866     // But without the subtraction to allow for INFINITY in comparison
07867     bool marginComparison(double lhs, double rhs, double margin) {
07868         return (lhs + margin >= rhs) && (rhs + margin >= lhs);
07869     }
07870
07871 }
07872
07873 namespace Catch {
07874     namespace Detail {
07875
07876         Approx::Approx ( double value )
07877         :   m_epsilon( std::numeric_limits<float>::epsilon()*100 ),
07878             m_margin( 0.0 ),
07879             m_scale( 0.0 ),
07880             m_value( value )
07881         {}
07882
07883         Approx Approx::custom() {
07884             return Approx( 0 );
07885         }
07886
07887         Approx Approx::operator-() const {
07888             auto temp(*this);
07889             temp.m_value = -temp.m_value;
07890             return temp;
07891         }
07892
07893         std::string Approx::toString() const {
07894             ReusableStringStream rss;
07895             rss << "Approx( " << ::Catch::Detail::stringify( m_value ) << " )";
07896             return rss.str();
07897         }
07898
07899         bool Approx::equalityComparisonImpl(const double other) const {
07900             // First try with fixed margin, then compute margin based on epsilon, scale and Approx's value
07901             // Thanks to Richard Harris for his help refining the scaled margin value
07902             return marginComparison(m_value, other, m_margin) ||
07903                marginComparison(m_value, other, m_epsilon * (m_scale + std::fabs(std::isinf(m_value)?
07904 0 : m_value)));
07905         }
07906
07907         void Approx::setMargin(double newMargin) {
07908             CATCH_ENFORCE(newMargin >= 0,
07909                 "Invalid Approx::margin: " << newMargin << ".");
07910             << " Approx::Margin has to be non-negative.");
07911             m_margin = newMargin;
07912         }
07913
07914         void Approx::setEpsilon(double newEpsilon) {
07915             CATCH_ENFORCE(newEpsilon >= 0 && newEpsilon <= 1.0,
07916                 "Invalid Approx::epsilon: " << newEpsilon << ".");
07917             << " Approx::epsilon has to be in [0, 1]");
07918             m_epsilon = newEpsilon;
07919         }
07920     } // end namespace Detail
07921
07922     namespace literals {
07923         Detail::Approx operator "" _a(long double val) {
07924             return Detail::Approx(val);
07925         }
07926         Detail::Approx operator "" _a(unsigned long long val) {
07927             return Detail::Approx(val);
07928         }
07929     } // end namespace literals
07930
07931     std::string StringMaker<Catch::Detail::Approx>::convert(Catch::Detail::Approx const& value) {

```

```

07932     return value.toString();
07933 }
07934
07935 } // end namespace Catch
07936 // end catch_approx.cpp
07937 // start catch_assertionhandler.cpp
07938
07939 // start catch_debugger.h
07940
07941 namespace Catch {
07942     bool isDebuggerActive();
07943 }
07944
07945 #ifdef CATCH_PLATFORM_MAC
07946
07947     #if defined(__i386__) || defined(__x86_64__)
07948         #define CATCH_TRAP() __asm__("int $3\n" : : ) /* NOLINT */
07949     #elif defined(__aarch64__)
07950         #define CATCH_TRAP() __asm__(".inst 0xd43e0000")
07951     #endif
07952
07953 #elif defined(CATCH_PLATFORM_IPHONE)
07954
07955     // use inline assembler
07956     #if defined(__i386__) || defined(__x86_64__)
07957         #define CATCH_TRAP() __asm__("int $3")
07958     #elif defined(__aarch64__)
07959         #define CATCH_TRAP() __asm__(".inst 0xd4200000")
07960     #elif defined(__arm__) && !defined(__thumb__)
07961         #define CATCH_TRAP() __asm__(".inst 0xe7f001f0")
07962     #elif defined(__arm__) && defined(__thumb__)
07963         #define CATCH_TRAP() __asm__(".inst 0xde01")
07964     #endif
07965
07966 #elif defined(CATCH_PLATFORM_LINUX)
07967     // If we can use inline assembler, do it because this allows us to break
07968     // directly at the location of the failing check instead of breaking inside
07969     // raise() called from it, i.e. one stack frame below.
07970     #if defined(__GNUC__) && (defined(__i386__) || defined(__x86_64__))
07971         #define CATCH_TRAP() asm volatile ("int $3") /* NOLINT */
07972     #else // Fall back to the generic way.
07973         #include <signal.h>
07974
07975         #define CATCH_TRAP() raise(SIGTRAP)
07976     #endif
07977 #elif defined(_MSC_VER)
07978     #define CATCH_TRAP() __debugbreak()
07979 #elif defined(__MINGW32__)
07980     extern "C" __declspec(dllimport) void __stdcall DebugBreak();
07981     #define CATCH_TRAP() DebugBreak()
07982 #endif
07983
07984 #ifndef CATCH_BREAK_INTO_DEBUGGER
07985     #ifdef CATCH_TRAP
07986         #define CATCH_BREAK_INTO_DEBUGGER() []{ if( Catch::isDebuggerActive() ) { CATCH_TRAP(); } }()
07987     #else
07988         #define CATCH_BREAK_INTO_DEBUGGER() []{}()
07989     #endif
07990 #endif
07991
07992 // end catch_debugger.h
07993 // start catch_run_context.h
07994
07995 // start catch_fatal_condition.h
07996
07997 #include <cassert>
07998
07999 namespace Catch {
08000
08001     // Wrapper for platform-specific fatal error (signals/SEH) handlers
08002     //
08003     // Tries to be cooperative with other handlers, and not step over
08004     // other handlers. This means that unknown structured exceptions
08005     // are passed on, previous signal handlers are called, and so on.
08006     //
08007     // Can only be instantiated once, and assumes that once a signal
08008     // is caught, the binary will end up terminating. Thus, there
08009     class FatalConditionHandler {
08010     public:
08011         bool m_started = false;
08012
08013         // Install/disengage implementation for specific platform.
08014         // Should be if-defed to work on current platform, can assume
08015         // engage-disengage 1:1 pairing.
08016         void engage_platform();
08017         void disengage_platform();
08018     public:
08019         // Should also have platform-specific implementations as needed

```

```

08019     FatalConditionHandler();
08020     ~FatalConditionHandler();
08021
08022     void engage() {
08023         assert(!m_started && "Handler cannot be installed twice.");
08024         m_started = true;
08025         engage_platform();
08026     }
08027
08028     void disengage() {
08029         assert(m_started && "Handler cannot be uninstalled without being installed first");
08030         m_started = false;
08031         disengage_platform();
08032     }
08033 };
08034
08035 class FatalConditionHandlerGuard {
08036     FatalConditionHandler* m_handler;
08037 public:
08038     FatalConditionHandlerGuard(FatalConditionHandler* handler):
08039         m_handler(handler) {
08040         m_handler->engage();
08041     }
08042     ~FatalConditionHandlerGuard() {
08043         m_handler->disengage();
08044     }
08045 };
08046 };
08047
08048 } // end namespace Catch
08049
08050 // end catch_fatal_condition.h
08051 #include <string>
08052
08053 namespace Catch {
08054     struct IMutableContext;
08055
08056     class RunContext : public IResultCapture, public IRunner {
08057 public:
08058         RunContext( RunContext const& ) = delete;
08059         RunContext& operator =( RunContext const& ) = delete;
08060
08061         explicit RunContext( IConfigPtr const& _config, IStreamingReporterPtr&& reporter );
08062         ~RunContext() override;
08063
08064         void testGroupStarting( std::string const& testSpec, std::size_t groupIndex, std::size_t
groupsCount );
08070         void testGroupEnded( std::string const& testSpec, Totals const& totals, std::size_t
groupIndex, std::size_t groupsCount );
08071
08072         Totals runTest(TestCase const& testCase);
08073
08074         IConfigPtr config() const;
08075         IStreamingReporter& reporter() const;
08076
08077     public: // IResultCapture
08078
08079         // Assertion handlers
08080         void handleExpr
08081             ( AssertionInfo const& info,
08082               ITransientExpression const& expr,
08083               AssertionReaction& reaction ) override;
08084         void handleMessage
08085             ( AssertionInfo const& info,
08086               ResultWas::OfType resultType,
08087               StringRef const& message,
08088               AssertionReaction& reaction ) override;
08089         void handleUnexpectedExceptionNotThrown
08090             ( AssertionInfo const& info,
08091               AssertionReaction& reaction ) override;
08092         void handleUnexpectedInflightException
08093             ( AssertionInfo const& info,
08094               std::string const& message,
08095               AssertionReaction& reaction ) override;
08096         void handleIncomplete
08097             ( AssertionInfo const& info ) override;
08098         void handleNonExpr
08099             ( AssertionInfo const& info,
08100               ResultWas::OfType resultType,
08101               AssertionReaction& reaction ) override;
08102
08103         bool sectionStarted( SectionInfo const& sectionInfo, Counts& assertions ) override;
08104
08105         void sectionEnded( SectionEndInfo const& endInfo ) override;

```



```

08106         void sectionEndedEarly( SectionEndInfo const& endInfo ) override;
08107
08108         auto acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo ) ->
    IGeneratorTracker& override;
08109
08110 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
08111     void benchmarkPreparing( std::string const& name ) override;
08112     void benchmarkStarting( BenchmarkInfo const& info ) override;
08113     void benchmarkEnded( BenchmarkStats<> const& stats ) override;
08114     void benchmarkFailed( std::string const& error ) override;
08115 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
08116
08117     void pushScopedMessage( MessageInfo const& message ) override;
08118     void popScopedMessage( MessageInfo const& message ) override;
08119
08120     void emplaceUnscopedMessage( MessageBuilder const& builder ) override;
08121
08122     std::string getCurrentTestName() const override;
08123
08124     const AssertionResult* getLastResult() const override;
08125
08126     void exceptionEarlyReported() override;
08127
08128     void handleFatalErrorCondition( StringRef message ) override;
08129
08130     bool lastAssertionPassed() override;
08131
08132     void assertionPassed() override;
08133
08134 public:
08135     // !TBD We need to do this another way!
08136     bool aborting() const final;
08137
08138 private:
08139
08140     void runCurrentTest( std::string& redirectedCout, std::string& redirectedCerr );
08141     void invokeActiveTestCase();
08142
08143     void resetAssertionInfo();
08144     bool testForMissingAssertions( Counts& assertions );
08145
08146     void assertionEnded( AssertionResult const& result );
08147     void reportExpr
08148         ( AssertionInfo const& info,
08149           ResultWas::OfType resultType,
08150           ITransientExpression const* expr,
08151           bool negated );
08152
08153     void populateReaction( AssertionReaction& reaction );
08154
08155 private:
08156
08157     void handleUnfinishedSections();
08158
08159     TestRunInfo m_runInfo;
08160     IMutableContext& m_context;
08161     TestCase const* m_activeTestCase = nullptr;
08162     ITracker* m_testCaseTracker = nullptr;
08163     Option<AssertionResult> m_lastResult;
08164
08165     IConfigPtr m_config;
08166     Totals m_totals;
08167     IStreamingReporterPtr m_reporter;
08168     std::vector<MessageInfo> m_messages;
08169     std::vector<ScopedMessage> m_messageScopes; /* Keeps owners of so-called unscoped messages. */
08170     AssertionInfo m_lastAssertionInfo;
08171     std::vector<SectionEndInfo> m_unfinishedSections;
08172     std::vector<ITracker*> m_activeSections;
08173     TrackerContext m_trackerContext;
08174     FatalConditionHandler m_fatalConditionhandler;
08175     bool m_lastAssertionPassed = false;
08176     bool m_shouldReportUnexpected = true;
08177     bool m_includeSuccessfulResults;
08178 };
08179
08180 void seedRng(IConfig const& config);
08181 unsigned int rngSeed();
08182 } // end namespace Catch
08183
08184 // end catch_run_context.h
08185 namespace Catch {
08186
08187     namespace {
08188         auto operator «( std::ostream& os, ITransientExpression const& expr ) -> std::ostream& {
08189             expr.streamReconstructedExpression( os );
08190             return os;
08191         }
08192     }

```

```

08192     }
08193
08194     LazyExpression::LazyExpression( bool isNegated )
08195     :   m_isNegated( isNegated )
08196     {}
08197
08198     LazyExpression::LazyExpression( LazyExpression const& other ) : m_isNegated( other.m_isNegated )
08199 {}
08200
08201     LazyExpression::operator bool() const {
08202         return m_transientExpression != nullptr;
08203     }
08204
08205     auto operator « ( std::ostream& os, LazyExpression const& lazyExpr ) -> std::ostream& {
08206         if( lazyExpr.m_isNegated )
08207             os « "!";
08208
08209         if( lazyExpr ) {
08210             if( lazyExpr.m_isNegated && lazyExpr.m_transientExpression->isBinaryExpression() )
08211                 os « "(" « *lazyExpr.m_transientExpression « ")";
08212             else
08213                 os « *lazyExpr.m_transientExpression;
08214         }
08215         else {
08216             os « "{** error - unchecked empty expression requested **}";
08217         }
08218         return os;
08219     }
08220
08221     AssertionHandler::AssertionHandler
08222     (   StringRef const& macroName,
08223         SourceLineInfo const& lineInfo,
08224         StringRef capturedExpression,
08225         ResultDisposition::Flags resultDisposition )
08226     :   m_assertionInfo{ macroName, lineInfo, capturedExpression, resultDisposition },
08227         m_resultCapture( getResultCapture() )
08228     {}
08229
08230     void AssertionHandler::handleExpr( ITransientExpression const& expr ) {
08231         m_resultCapture.handleExpr( m_assertionInfo, expr, m_reaction );
08232     }
08233     void AssertionHandler::handleMessage(ResultWas::OfType resultType, StringRef const& message) {
08234         m_resultCapture.handleMessage( m_assertionInfo, resultType, message, m_reaction );
08235     }
08236
08237     auto AssertionHandler::allowThrows() const -> bool {
08238         return getCurrentContext().getConfig()->allowThrows();
08239     }
08240
08241     void AssertionHandler::complete() {
08242         setCompleted();
08243         if( m_reaction.shouldDebugBreak ) {
08244             // If you find your debugger stopping you here then go one level up on the
08245             // call-stack for the code that caused it (typically a failed assertion)
08246
08247             // (To go back to the test and change execution, jump over the throw, next)
08248             CATCH_BREAK_INTO_DEBUGGER();
08249         }
08250         if (m_reaction.shouldThrow) {
08251             #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
08252                 throw Catch::TestFailureException();
08253             #else
08254                 CATCH_ERROR( "Test failure requires aborting test!" );
08255             #endif
08256         }
08257     }
08258     void AssertionHandler::setCompleted() {
08259         m_completed = true;
08260     }
08261
08262     void AssertionHandler::handleUnexpectedInflightException() {
08263         m_resultCapture.handleUnexpectedInflightException( m_assertionInfo,
08264             Catch::translateActiveException(), m_reaction );
08265     }
08266     void AssertionHandler::handleExceptionThrownAsExpected() {
08267         m_resultCapture.handleNonExpr(m_assertionInfo, ResultWas::Ok, m_reaction);
08268     }
08269     void AssertionHandler::handleExceptionNotThrownAsExpected() {
08270         m_resultCapture.handleNonExpr(m_assertionInfo, ResultWas::Ok, m_reaction);
08271     }
08272
08273     void AssertionHandler::handleUnexpectedExceptionNotThrown() {
08274         m_resultCapture.handleUnexpectedExceptionNotThrown( m_assertionInfo, m_reaction );
08275     }
08276

```

```

08277     void AssertionHandler::handleThrowingCallSkipped() {
08278         m_resultCapture.handleNonExpr(m_assertionInfo, ResultWas::Ok, m_reaction);
08279     }
08280
08281     // This is the overload that takes a string and infers the Equals matcher from it
08282     // The more general overload, that takes any string matcher, is in catch_capture_matchers.cpp
08283     void handleExceptionMatchExpr( AssertionHandler& handler, std::string const& str, StringRef const&
matcherString ) {
08284         handleExceptionMatchExpr( handler, Matchers::Equals( str ), matcherString );
08285     }
08286
08287 } // namespace Catch
08288 // end catch_assertionhandler.cpp
08289 // start catch_assertionresult.cpp
08290
08291 namespace Catch {
08292     AssertionResultData::AssertionResultData(ResultWas::OfType _resultType, LazyExpression const&
_lazyExpression):
08293         lazyExpression(_lazyExpression),
08294         resultType(_resultType) {}
08295
08296     std::string AssertionResultData::reconstructExpression() const {
08297
08298         if( reconstructedExpression.empty() ) {
08299             if( lazyExpression ) {
08300                 ReusableStringStream rss;
08301                 rss << lazyExpression;
08302                 reconstructedExpression = rss.str();
08303             }
08304         }
08305         return reconstructedExpression;
08306     }
08307
08308     AssertionResult::AssertionResult( AssertionInfo const& info, AssertionResultData const& data )
08309     :   m_info( info ),
08310         m_resultData( data )
08311     {}
08312
08313     // Result was a success
08314     bool AssertionResult::succeeded() const {
08315         return Catch::isOk( m_resultData.resultType );
08316     }
08317
08318     // Result was a success, or failure is suppressed
08319     bool AssertionResult::isOk() const {
08320         return Catch::isOk( m_resultData.resultType ) || shouldSuppressFailure(
m_info.resultDisposition );
08321     }
08322
08323     ResultWas::OfType AssertionResult::getResultType() const {
08324         return m_resultData.resultType;
08325     }
08326
08327     bool AssertionResult::hasExpression() const {
08328         return !m_info.capturedExpression.empty();
08329     }
08330
08331     bool AssertionResult::hasMessage() const {
08332         return !m_resultData.message.empty();
08333     }
08334
08335     std::string AssertionResult::getExpression() const {
08336         // Possibly overallocating by 3 characters should be basically free
08337         std::string expr; expr.reserve(m_info.capturedExpression.size() + 3);
08338         if (isFalseTest(m_info.resultDisposition)) {
08339             expr += "!(";
08340         }
08341         expr += m_info.capturedExpression;
08342         if (isFalseTest(m_info.resultDisposition)) {
08343             expr += ')';
08344         }
08345         return expr;
08346     }
08347
08348     std::string AssertionResult::getExpressionInMacro() const {
08349         std::string expr;
08350         if( m_info.macroName.empty() )
08351             expr = static_cast<std::string>(m_info.capturedExpression);
08352         else {
08353             expr.reserve( m_info.macroName.size() + m_info.capturedExpression.size() + 4 );
08354             expr += m_info.macroName;
08355             expr += " ( ";
08356             expr += m_info.capturedExpression;
08357             expr += " )";
08358         }
08359         return expr;
08360     }

```

```

08361
08362     bool AssertionResult::hasExpandedExpression() const {
08363         return hasExpression() && getExpandedExpression() != getExpression();
08364     }
08365
08366     std::string AssertionResult::getExpandedExpression() const {
08367         std::string expr = m_resultData.reconstructExpression();
08368         return expr.empty()
08369             ? getExpression()
08370             : expr;
08371     }
08372
08373     std::string AssertionResult::getMessage() const {
08374         return m_resultData.message;
08375     }
08376     SourceLineInfo AssertionResult::getSourceInfo() const {
08377         return m_info.lineInfo;
08378     }
08379
08380     StringRef AssertionResult::getTestMacroName() const {
08381         return m_info.macroName;
08382     }
08383 } // end namespace Catch
08384 // end catch_assertionresult.cpp
08385 // start catch_capture_matchers.cpp
08386 namespace Catch {
08387     using StringMatcher = Matchers::Impl::MatcherBase<std::string>;
08388
08389     // This is the general overload that takes a any string matcher
08390     // There is another overload, in catch_assertionhandler.h/.cpp, that only takes a string and
08391     // infers
08392     // the Equals matcher (so the header does not mention matchers)
08393     void handleExceptionMatchExpr( AssertionHandler& handler, StringMatcher const& matcher, StringRef
08394     const& matcherString ) {
08395         std::string exceptionMessage = Catch::translateActiveException();
08396         MatchExpr<std::string, StringMatcher const&> expr( exceptionMessage, matcher, matcherString );
08397         handler.handleExpr( expr );
08398     }
08399 }
08400
08401 } // namespace Catch
08402 // end catch_capture_matchers.cpp
08403 // start catch_commandline.cpp
08404
08405 // start catch_commandline.h
08406
08407 // start catch_clara.h
08408
08409 // Use Catch's value for console width (store Clara's off to the side, if present)
08410 #ifdef CLARA_CONFIG_CONSOLE_WIDTH
08411 #define CATCH_TEMP_CLARA_CONFIG_CONSOLE_WIDTH CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH
08412 #undef CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH
08413 #endif
08414 #define CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH CATCH_CONFIG_CONSOLE_WIDTH-1
08415
08416 #ifdef __clang__
08417 #pragma clang diagnostic push
08418 #pragma clang diagnostic ignored "-Wweak-vtables"
08419 #pragma clang diagnostic ignored "-Wexit-time-destructors"
08420 #pragma clang diagnostic ignored "-Wshadow"
08421 #endif
08422
08423 // start clara.hpp
08424 // Copyright 2017 Two Blue Cubes Ltd. All rights reserved.
08425 //
08426 // Distributed under the Boost Software License, Version 1.0. (See accompanying
08427 // file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
08428 //
08429 // See https://github.com/philsquared/Clara for more details
08430
08431 // Clara v1.1.5
08432
08433 #ifndef CATCH_CLARA_CONFIG_CONSOLE_WIDTH
08434 #define CATCH_CLARA_CONFIG_CONSOLE_WIDTH 80
08435 #endif
08436 #ifndef CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH
08437 #define CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH CATCH_CLARA_CONFIG_CONSOLE_WIDTH
08438 #endif
08439 #ifndef CLARA_CONFIG_OPTIONAL_TYPE
08440 #define CLARA_CONFIG_OPTIONAL_TYPE optional
08441 #endif
08442 #ifndef CLARA_CONFIG_OPTIONAL_TYPE
08443 #define CLARA_CONFIG_OPTIONAL_TYPE optional
08444 #endif
08445 #include <optional>

```

```

08446 #define CLARA_CONFIG_OPTIONAL_TYPE std::optional
08447 #endif
08448 #endif
08449 #endif
08450
08451 // ----- #included from clara_textflow.hpp -----
08452
08453 // TextFlowCpp
08454 //
08455 // A single-header library for wrapping and laying out basic text, by Phil Nash
08456 //
08457 // Distributed under the Boost Software License, Version 1.0. (See accompanying
08458 // file LICENSE.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
08459 //
08460 // This project is hosted at https://github.com/philsquared/textflowcpp
08461
08462
08463 #include <cassert>
08464 #include <ostream>
08465 #include <sstream>
08466 #include <vector>
08467
08468 #ifndef CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH
08469 #define CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH 80
08470 #endif
08471
08472 namespace Catch {
08473     namespace clara {
08474         namespace TextFlow {
08475
08476             inline auto isWhitespace(char c) -> bool {
08477                 static std::string chars = "\t\n\r";
08478                 return chars.find(c) != std::string::npos;
08479             }
08480
08481             inline auto isBreakableBefore(char c) -> bool {
08482                 static std::string chars = "[(|<";
08483                 return chars.find(c) != std::string::npos;
08484             }
08485
08486             inline auto isBreakableAfter(char c) -> bool {
08487                 static std::string chars = "])>.,;:*-=&/\\\"";
08488                 return chars.find(c) != std::string::npos;
08489             }
08490
08491             class Columns;
08492
08493             class Column {
08494             public:
08495                 std::vector<std::string> m_strings;
08496                 size_t m_width = CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH;
08497                 size_t m_indent = 0;
08498                 size_t m_initialIndent = std::string::npos;
08499
08500                 class iterator {
08501                 public:
08502                     friend Column;
08503
08504                     Column const& m_column;
08505                     size_t m_stringIndex = 0;
08506                     size_t m_pos = 0;
08507
08508                     size_t m_len = 0;
08509                     size_t m_end = 0;
08510                     bool m_suffix = false;
08511
08512                     iterator(Column const& column, size_t stringIndex)
08513                         : m_column(column),
08514                         m_stringIndex(stringIndex) {}
08514
08515                     auto line() const -> std::string const& { return m_column.m_strings[m_stringIndex]; }
08516
08517                     auto isBoundary(size_t at) const -> bool {
08518                         assert(at > 0);
08519                         assert(at <= line().size());
08520
08521                         return at == line().size() ||
08522                             (isWhitespace(line()[at]) && !isWhitespace(line()[at - 1])) ||
08523                             isBreakableBefore(line()[at]) ||
08524                             isBreakableAfter(line()[at - 1]);
08525                     }
08526
08527                     void calcLength() {
08528                         assert(m_stringIndex < m_column.m_strings.size());
08529
08530                         m_suffix = false;
08531                         auto width = m_column.m_width - m_indent;
08532                         m_end = m_pos;
08533
08534                         if (line()[m_pos] == '\n') {
08535                             ++m_end;
08536                         }
08537                     }
08538
08539                     size_t len() const { return m_len; }
08540                     size_t end() const { return m_end; }
08541                     bool suffix() const { return m_suffix; }
08542
08543                     operator std::string const&() const {
08544                         calcLength();
08545                         if (m_pos < m_end)
08546                             return line().substr(m_pos, m_end - m_pos);
08547                         return line().substr(m_pos);
08548                     }
08549
08550                     &operator+=(std::string const& str) {
08551                         m_column.m_strings[m_stringIndex] += str;
08552                         return *this;
08553                     }
08554                 };
08555
08556                 iterator begin() const { return iterator(*this, 0); }
08557                 iterator end() const { return iterator(*this, m_strings.size()); }
08558
08559                 Column& operator+=(Column const& rhs) {
08560                     m_strings.insert(m_strings.end(), rhs.m_strings.begin(), rhs.m_strings.end());
08561                     return *this;
08562                 }
08563
08564                 size_t width() const { return m_width; }
08565                 size_t indent() const { return m_indent; }
08566                 Column& indent(size_t indent) {
08567                     if (m_indent != indent)
08568                         m_initialIndent = std::string::npos;
08569                     m_indent = indent;
08570                     return *this;
08571                 }
08572
08573                 Column& operator+=(std::string const& str) {
08574                     return *this += Column(str);
08575                 }
08576
08577                 Column& operator+=(Column const& column) {
08578                     if (m_initialIndent == std::string::npos)
08579                         m_indent = column.m_indent;
08580
08581                     column.m_strings.insert(column.m_strings.begin(), m_strings.begin(), m_strings.end());
08582                     m_strings = column.m_strings;
08583                     m_width = column.m_width;
08584                     return *this;
08585                 }
08586
08587                 Column& operator=(Column const& rhs) {
08588                     m_strings = rhs.m_strings;
08589                     m_width = rhs.m_width;
08590                     m_indent = rhs.m_indent;
08591                     m_initialIndent = rhs.m_initialIndent;
08592                     return *this;
08593                 }
08594             };
08595
08596             Columns operator+(Column const& lhs, Column const& rhs) {
08597                 Columns cols;
08598                 cols += lhs;
08599                 cols += rhs;
08600                 return cols;
08601             }
08602
08603             Columns operator+(Column const& column, std::string const& str) {
08604                 return column += Column(str);
08605             }
08606
08607             Columns operator+(Column const& column, Column const& rhs) {
08608                 return column += rhs;
08609             }
08610
08611             Columns& operator+=(Columns const& rhs) {
08612                 *this += Column{};
08613                 *this += rhs;
08614                 return *this;
08615             }
08616
08617             Columns& operator+=(std::string const& str) {
08618                 *this += Column(str);
08619                 return *this;
08620             }
08621
08622             Columns& indent(size_t indent) {
08623                 *this += Column{};
08624                 *this += Column(indent);
08625                 return *this;
08626             }
08627
08628             auto operator+(std::string const& str) { return Column{} + Column(str); }
08629             auto operator+(Column const& column) { return Column{} + column; }
08630
08631             auto operator+(std::string const& lhs, std::string const& rhs) { return Column(lhs) + Column(rhs); }
08632             auto operator+(std::string const& lhs, Column const& rhs) { return Column(lhs) + rhs; }
08633             auto operator+(Column const& lhs, std::string const& rhs) { return lhs + Column(rhs); }
08634             auto operator+(Column const& lhs, Column const& rhs) { return lhs + rhs; }
08635
08636             Columns& operator+=(std::string const& str) {
08637                 *this += Column(str);
08638                 return *this;
08639             }
08640
08641             Columns& indent(size_t indent) {
08642                 *this += Column(indent);
08643                 return *this;
08644             }
08645
08646             auto operator+=(std::string const& str) {
08647                 *this += Column(str);
08648                 return *this;
08649             }
08650
08651             auto operator+=(Column const& column) {
08652                 *this += column;
08653                 return *this;
08654             }
08655
08656             auto operator+=(Columns const& rhs) {
08657                 *this += Column{};
08658                 *this += rhs;
08659                 return *this;
08660             }
08661
08662             auto operator+=(Columns const& rhs) const {
08663                 *this += Column{};
08664                 *this += rhs;
08665                 return *this;
08666             }
08667
08668             auto operator+(Columns const& rhs) const {
08669                 *this += Column{};
08670                 *this += rhs;
08671                 return *this;
08672             }
08673
08674             Columns& operator=(Columns const& rhs) {
08675                 *this += Column{};
08676                 *this += rhs;
08677                 return *this;
08678             }
08679
08680             Columns& operator=(std::string const& str) {
08681                 *this += Column(str);
08682                 return *this;
08683             }
08684
08685             Columns& operator=(Column const& rhs) {
08686                 *this += Column{};
08687                 *this += rhs;
08688                 return *this;
08689             }
08690
08691             Columns& operator=(std::string const& lhs, std::string const& rhs) {
08692                 *this += Column(lhs) + Column(rhs);
08693                 return *this;
08694             }
08695
08696             Columns& operator=(std::string const& lhs, Column const& rhs) {
08697                 *this += Column(lhs) + rhs;
08698                 return *this;
08699             }
08700
08701             Columns& operator=(Column const& lhs, std::string const& rhs) {
08702                 *this += lhs + Column(rhs);
08703                 return *this;
08704             }
08705
08706             Columns& operator=(Column const& lhs, Column const& rhs) {
08707                 *this += lhs + rhs;
08708                 return *this;
08709             }
08710
08711             Columns& operator=(Columns const& rhs) {
08712                 *this += Column{};
08713                 *this += rhs;
08714                 return *this;
08715             }
08716
08717             Columns& operator=(Columns const& rhs) const {
08718                 *this += Column{};
08719                 *this += rhs;
08720                 return *this;
08721             }
08722
08723             auto operator+(Columns const& rhs) const {
08724                 *this += Column{};
08725                 *this += rhs;
08726                 return *this;
08727             }
08728
08729             Columns& operator+=(Columns const& rhs) const {
08730                 *this += Column{};
08731                 *this += rhs;
08732                 return *this;
08733             }
08734
08735             Columns& operator+=(Columns const& rhs) const {
08736                 *this += Column{};
08737                 *this += rhs;
08738                 return *this;
08739             }
08740
08741             Columns& operator+=(Columns const& rhs) const {
08742                 *this += Column{};
08743                 *this += rhs;
08744                 return *this;
08745             }
08746
08747             Columns& operator+=(Columns const& rhs) const {
08748                 *this += Column{};
08749                 *this += rhs;
08750                 return *this;
08751             }
08752
08753             Columns& operator+=(Columns const& rhs) const {
08754                 *this += Column{};
08755                 *this += rhs;
08756                 return *this;
08757             }
08758
08759             Columns& operator+=(Columns const& rhs) const {
08760                 *this += Column{};
08761                 *this += rhs;
08762                 return *this;
08763             }
08764
08765             Columns& operator+=(Columns const& rhs) const {
08766                 *this += Column{};
08
```

```

08533         }
08534         while (m_end < line().size() && line()[m_end] != '\n')
08535             ++m_end;
08536
08537         if (m_end < m_pos + width) {
08538             m_len = m_end - m_pos;
08539         } else {
08540             size_t len = width;
08541             while (len > 0 && !isBoundary(m_pos + len))
08542                 --len;
08543             while (len > 0 && isWhitespace(line()[m_pos + len - 1]))
08544                 --len;
08545
08546             if (len > 0) {
08547                 m_len = len;
08548             } else {
08549                 m_suffix = true;
08550                 m_len = width - 1;
08551             }
08552         }
08553     }
08554
08555     auto indent() const -> size_t {
08556         auto initial = m_pos == 0 && m_stringIndex == 0 ? m_column.m_initialIndent :
std::string::npos;
08557         return initial == std::string::npos ? m_column.m_indent : initial;
08558     }
08559
08560     auto addIndentAndSuffix(std::string const &plain) const -> std::string {
08561         return std::string(indent(), ' ') + (m_suffix ? plain + "-" : plain);
08562     }
08563
08564     public:
08565         using difference_type = std::ptrdiff_t;
08566         using value_type = std::string;
08567         using pointer = value_type * ;
08568         using reference = value_type & ;
08569         using iterator_category = std::forward_iterator_tag;
08570
08571         explicit iterator(Column const& column) : m_column(column) {
08572             assert(m_column.m_width > m_column.m_indent);
08573             assert(m_column.m_initialIndent == std::string::npos || m_column.m_width >
m_column.m_initialIndent);
08574             calcLength();
08575             if (m_len == 0)
08576                 m_stringIndex++; // Empty string
08577         }
08578
08579         auto operator *() const -> std::string {
08580             assert(m_stringIndex < m_column.m_strings.size());
08581             assert(m_pos <= m_end);
08582             return addIndentAndSuffix(line().substr(m_pos, m_len));
08583         }
08584
08585         auto operator ++() -> iterator& {
08586             m_pos += m_len;
08587             if (m_pos < line().size() && line()[m_pos] == '\n')
08588                 m_pos += 1;
08589             else
08590                 while (m_pos < line().size() && isWhitespace(line()[m_pos]))
08591                     ++m_pos;
08592
08593             if (m_pos == line().size()) {
08594                 m_pos = 0;
08595                 ++m_stringIndex;
08596             }
08597             if (m_stringIndex < m_column.m_strings.size())
08598                 calcLength();
08599             return *this;
08600         }
08601         auto operator ++(int) -> iterator {
08602             iterator prev(*this);
08603             operator ++();
08604             return prev;
08605         }
08606
08607         auto operator ==(iterator const& other) const -> bool {
08608             return
08609                 m_pos == other.m_pos &&
08610                 m_stringIndex == other.m_stringIndex &&
08611                 &m_column == &other.m_column;
08612         }
08613         auto operator !=(iterator const& other) const -> bool {
08614             return !operator==(other);
08615         }
08616     };
08617     using const_iterator = iterator;

```

```

08618
08619     explicit Column(std::string const& text) { m_strings.push_back(text); }
08620
08621     auto width(size_t newWidth) -> Column& {
08622         assert(newWidth > 0);
08623         m_width = newWidth;
08624         return *this;
08625     }
08626     auto indent(size_t newIndent) -> Column& {
08627         m_indent = newIndent;
08628         return *this;
08629     }
08630     auto initialIndent(size_t newIndent) -> Column& {
08631         m_initialIndent = newIndent;
08632         return *this;
08633     }
08634
08635     auto width() const -> size_t { return m_width; }
08636     auto begin() const -> iterator { return iterator(*this); }
08637     auto end() const -> iterator { return { *this, m_strings.size() }; }
08638
08639     inline friend std::ostream& operator << (std::ostream& os, Column const& col) {
08640         bool first = true;
08641         for (auto line : col) {
08642             if (first)
08643                 first = false;
08644             else
08645                 os << "\n";
08646             os << line;
08647         }
08648         return os;
08649     }
08650
08651     auto operator + (Column const& other)->Columns;
08652
08653     auto toString() const -> std::string {
08654         std::ostringstream oss;
08655         oss << *this;
08656         return oss.str();
08657     }
08658 };
08659
08660 class Spacer : public Column {
08661 public:
08662     explicit Spacer(size_t spaceWidth) : Column("") {
08663         width(spaceWidth);
08664     }
08665 };
08666
08667 class Columns {
08668     std::vector<Column> m_columns;
08669 public:
08670     class iterator {
08671     friend Columns;
08672     struct EndTag {};
08673
08674     std::vector<Column> const& m_columns;
08675     std::vector<Column::iterator> m_iterators;
08676     size_t m_activeIterators;
08677
08678     iterator(Columns const& columns, EndTag)
08679         : m_columns(columns.m_columns),
08680         m_activeIterators(0) {
08681         m_iterators.reserve(m_columns.size());
08682
08683         for (auto const& col : m_columns)
08684             m_iterators.push_back(col.end());
08685     }
08686
08687 public:
08688     using difference_type = std::ptrdiff_t;
08689     using value_type = std::string;
08690     using pointer = value_type * ;
08691     using reference = value_type & ;
08692     using iterator_category = std::forward_iterator_tag;
08693
08694     explicit iterator(Columns const& columns)
08695         : m_columns(columns.m_columns),
08696         m_activeIterators(m_columns.size()) {
08697         m_iterators.reserve(m_columns.size());
08698
08699         for (auto const& col : m_columns)
08700             m_iterators.push_back(col.begin());
08701     }
08702
08703
08704

```

```

08705
08706     auto operator ==(iterator const& other) const -> bool {
08707         return m_iterators == other.m_iterators;
08708     }
08709     auto operator !=(iterator const& other) const -> bool {
08710         return m_iterators != other.m_iterators;
08711     }
08712     auto operator *() const -> std::string {
08713         std::string row, padding;
08714
08715         for (size_t i = 0; i < m_columns.size(); ++i) {
08716             auto width = m_columns[i].width();
08717             if (m_iterators[i] != m_columns[i].end()) {
08718                 std::string col = *m_iterators[i];
08719                 row += padding + col;
08720                 if (col.size() < width)
08721                     padding = std::string(width - col.size(), ' ');
08722                 else
08723                     padding = "";
08724             } else {
08725                 padding += std::string(width, ' ');
08726             }
08727         }
08728         return row;
08729     }
08730     auto operator ++() -> iterator& {
08731         for (size_t i = 0; i < m_columns.size(); ++i) {
08732             if (m_iterators[i] != m_columns[i].end())
08733                 ++m_iterators[i];
08734         }
08735         return *this;
08736     }
08737     auto operator ++(int) -> iterator {
08738         iterator prev(*this);
08739         operator ++();
08740         return prev;
08741     }
08742 };
08743 using const_iterator = iterator;
08744
08745 auto begin() const -> iterator { return iterator(*this); }
08746 auto end() const -> iterator { return { *this, iterator::EndTag() }; }
08747
08748 auto operator += (Column const& col) -> Columns& {
08749     m_columns.push_back(col);
08750     return *this;
08751 }
08752 auto operator + (Column const& col) -> Columns {
08753     Columns combined = *this;
08754     combined += col;
08755     return combined;
08756 }
08757
08758 inline friend std::ostream& operator << (std::ostream& os, Columns const& cols) {
08759
08760     bool first = true;
08761     for (auto line : cols) {
08762         if (first)
08763             first = false;
08764         else
08765             os << "\n";
08766         os << line;
08767     }
08768     return os;
08769 }
08770
08771 auto toString() const -> std::string {
08772     std::ostringstream oss;
08773     oss << *this;
08774     return oss.str();
08775 }
08776 };
08777
08778 inline auto Column::operator + (Column const& other) -> Columns {
08779     Columns cols;
08780     cols += *this;
08781     cols += other;
08782     return cols;
08783 }
08784 }
08785
08786 }
08787 }
08788
08789 // ----- end of #include from clara_textflow.hpp -----
08790 // ..... back in clara.hpp
08791

```



```

08792 #include <cctype>
08793 #include <string>
08794 #include <memory>
08795 #include <set>
08796 #include <algorithm>
08797
08798 #if !defined(CATCH_PLATFORM_WINDOWS) && ( defined(WIN32) || defined(__WIN32__) || defined(_WIN32) ||
    defined(_MSC_VER) )
08799 #define CATCH_PLATFORM_WINDOWS
08800 #endif
08801
08802 namespace Catch { namespace clara {
08803 namespace detail {
08804
08805     // Traits for extracting arg and return type of lambdas (for single argument lambdas)
08806     template<typename L>
08807     struct UnaryLambdaTraits : UnaryLambdaTraits<decltype( &L::operator() )> {};
08808
08809     template<typename ClassT, typename ReturnT, typename... Args>
08810     struct UnaryLambdaTraits<ReturnT( ClassT::* )( Args... ) const> {
08811         static const bool isValid = false;
08812     };
08813
08814     template<typename ClassT, typename ReturnT, typename ArgT>
08815     struct UnaryLambdaTraits<ReturnT( ClassT::* )( ArgT ) const> {
08816         static const bool isValid = true;
08817         using ArgType = typename std::remove_const<typename std::remove_reference<ArgT>::type>::type;
08818         using ReturnType = ReturnT;
08819     };
08820
08821     class TokenStream;
08822
08823     // Transport for raw args (copied from main args, or supplied via init list for testing)
08824     class Args {
08825     friend TokenStream;
08826     std::string m_exeName;
08827     std::vector<std::string> m_args;
08828
08829     public:
08830         Args( int argc, char const* const* argv )
08831             : m_exeName(argv[0]),
08832               m_args(argv + 1, argv + argc) {}
08833
08834         Args( std::initializer_list<std::string> args )
08835             : m_exeName( *args.begin() ),
08836               m_args( args.begin()+1, args.end() )
08837         {}
08838
08839         auto exeName() const -> std::string {
08840             return m_exeName;
08841         }
08842     };
08843
08844     // Wraps a token coming from a token stream. These may not directly correspond to strings as a
    single string
08845     // may encode an option + its argument if the : or = form is used
08846     enum class TokenType {
08847         Option, Argument
08848     };
08849     struct Token {
08850         TokenType type;
08851         std::string token;
08852     };
08853
08854     inline auto isOptPrefix( char c ) -> bool {
08855         return c == '-' ||
08856 #ifdef CATCH_PLATFORM_WINDOWS
08857             || c == '/'
08858 #endif
08859         ;
08860     }
08861
08862     // Abstracts iterators into args as a stream of tokens, with option arguments uniformly handled
08863     class TokenStream {
08864     using Iterator = std::vector<std::string>::const_iterator;
08865     Iterator it;
08866     Iterator itEnd;
08867     std::vector<Token> m_tokenBuffer;
08868
08869     void loadBuffer() {
08870         m_tokenBuffer.resize( 0 );
08871
08872         // Skip any empty strings
08873         while( it != itEnd && it->empty() )
08874             ++it;
08875
08876         if( it != itEnd ) {

```

```

08877         auto const &next = *it;
08878         if( isOptPrefix( next[0] ) ) {
08879             auto delimiterPos = next.find_first_of( " :=" );
08880             if( delimiterPos != std::string::npos ) {
08881                 m_tokenBuffer.push_back( { TokenType::Option, next.substr( 0, delimiterPos ) }
08882 );
08883                 m_tokenBuffer.push_back( { TokenType::Argument, next.substr( delimiterPos + 1
08884 ) } );
08885             } else {
08886                 if( next[1] != '-' && next.size() > 2 ) {
08887                     std::string opt = "- ";
08888                     for( size_t i = 1; i < next.size(); ++i ) {
08889                         opt[i] = next[i];
08890                         m_tokenBuffer.push_back( { TokenType::Option, opt } );
08891                     }
08892                 } else {
08893                     m_tokenBuffer.push_back( { TokenType::Option, next } );
08894                 }
08895             } else {
08896                 m_tokenBuffer.push_back( { TokenType::Argument, next } );
08897             }
08898         }
08899     }
08900 public:
08901     explicit TokenStream( Args const &args ) : TokenStream( args.m_args.begin(), args.m_args.end()
08902 ) {}
08903     TokenStream( Iterator it, Iterator itEnd ) : it( it ), itEnd( itEnd ) {
08904         loadBuffer();
08905     }
08906     explicit operator bool() const {
08907         return !m_tokenBuffer.empty() || it != itEnd;
08908     }
08909     auto count() const -> size_t { return m_tokenBuffer.size() + (itEnd - it); }
08910     auto operator*() const -> Token {
08911         assert( !m_tokenBuffer.empty() );
08912         return m_tokenBuffer.front();
08913     }
08914     auto operator->() const -> Token const * {
08915         assert( !m_tokenBuffer.empty() );
08916         return &m_tokenBuffer.front();
08917     }
08918     auto operator++() -> TokenStream & {
08919         if( m_tokenBuffer.size() >= 2 ) {
08920             m_tokenBuffer.erase( m_tokenBuffer.begin() );
08921         } else {
08922             if( it != itEnd )
08923                 ++it;
08924             loadBuffer();
08925         }
08926         return *this;
08927     }
08928 };
08929
08930 class ResultBase {
08931 public:
08932     enum Type {
08933         Ok, LogicError, RuntimeError
08934     };
08935 protected:
08936     ResultBase( Type type ) : m_type( type ) {}
08937     virtual ~ResultBase() = default;
08938     virtual void enforceOk() const = 0;
08939     Type m_type;
08940 };
08941
08942 template<typename T>
08943 class ResultValueBase : public ResultBase {
08944 public:
08945     auto value() const -> T const & {
08946         enforceOk();
08947         return m_value;
08948     }
08949 protected:
08950     ResultValueBase( Type type ) : ResultBase( type ) {}
08951

```

```

08961     ResultValueBase( ResultValueBase const &other ) : ResultBase( other ) {
08962         if( m_type == ResultBase::Ok )
08963             new( &m_value ) T( other.m_value );
08964     }
08965
08966     ResultValueBase( Type, T const &value ) : ResultBase( Ok ) {
08967         new( &m_value ) T( value );
08968     }
08969
08970     auto operator=( ResultValueBase const &other ) -> ResultValueBase & {
08971         if( m_type == ResultBase::Ok )
08972             m_value.~T();
08973         ResultBase::operator=(other);
08974         if( m_type == ResultBase::Ok )
08975             new( &m_value ) T( other.m_value );
08976         return *this;
08977     }
08978
08979     ~ResultValueBase() override {
08980         if( m_type == Ok )
08981             m_value.~T();
08982     }
08983
08984     union {
08985         T m_value;
08986     };
08987 };
08988
08989 template<>
08990 class ResultValueBase<void> : public ResultBase {
08991 protected:
08992     using ResultBase::ResultBase;
08993 };
08994
08995 template<typename T = void>
08996 class BasicResult : public ResultValueBase<T> {
08997 public:
08998     template<typename U>
08999     explicit BasicResult( BasicResult<U> const &other )
09000     :   ResultValueBase<T>( other.type() ),
09001         m_errorMessage( other.errorMessage() )
09002     {
09003         assert( type() != ResultBase::Ok );
09004     }
09005
09006     template<typename U>
09007     static auto ok( U const &value ) -> BasicResult { return { ResultBase::Ok, value }; }
09008     static auto ok() -> BasicResult { return { ResultBase::Ok }; }
09009     static auto logicError( std::string const &message ) -> BasicResult { return {
ResultBase::LogicError, message }; }
09010     static auto runtimeError( std::string const &message ) -> BasicResult { return {
ResultBase::RuntimeError, message }; }
09011
09012     explicit operator bool() const { return m_type == ResultBase::Ok; }
09013     auto type() const -> ResultBase::Type { return m_type; }
09014     auto errorMessage() const -> std::string { return m_errorMessage; }
09015
09016 protected:
09017     void enforceOk() const override {
09018
09019         // Errors shouldn't reach this point, but if they do
09020         // the actual error message will be in m_errorMessage
09021         assert( m_type != ResultBase::LogicError );
09022         assert( m_type != ResultBase::RuntimeError );
09023         if( m_type != ResultBase::Ok )
09024             std::abort();
09025     }
09026
09027     std::string m_errorMessage; // Only populated if resultType is an error
09028
09029     BasicResult( ResultBase::Type type, std::string const &message )
09030     :   ResultValueBase<T>(type),
09031         m_errorMessage(message)
09032     {
09033         assert( m_type != ResultBase::Ok );
09034     }
09035
09036     using ResultValueBase<T>::ResultValueBase;
09037     using ResultBase::m_type;
09038 };
09039
09040 enum class ParseResultType {
09041     Matched, NoMatch, ShortCircuitAll, ShortCircuitSame
09042 };
09043
09044 class ParseState {
09045 public:

```

```

09046
09047     ParseState( ParseResultType type, TokenStream const &remainingTokens )
09048     : m_type(type),
09049       m_remainingTokens( remainingTokens )
09050     {}
09051
09052     auto type() const -> ParseResultType { return m_type; }
09053     auto remainingTokens() const -> TokenStream { return m_remainingTokens; }
09054
09055 private:
09056     ParseResultType m_type;
09057     TokenStream m_remainingTokens;
09058 };
09059
09060 using Result = BasicResult<void>;
09061 using ParserResult = BasicResult<ParseResultType>;
09062 using InternalParserResult = BasicResult<ParseState>;
09063
09064 struct HelpColumns {
09065     std::string left;
09066     std::string right;
09067 };
09068
09069 template<typename T>
09070 inline auto convertInto( std::string const &source, T& target ) -> ParserResult {
09071     std::stringstream ss;
09072     ss << source;
09073     ss >> target;
09074     if( ss.fail() )
09075         return ParserResult::runtimeError( "Unable to convert '" + source + "' to destination
type" );
09076     else
09077         return ParserResult::ok( ParseResultType::Matched );
09078 }
09079 inline auto convertInto( std::string const &source, std::string& target ) -> ParserResult {
09080     target = source;
09081     return ParserResult::ok( ParseResultType::Matched );
09082 }
09083 inline auto convertInto( std::string const &source, bool &target ) -> ParserResult {
09084     std::string srcLC = source;
09085     std::transform( srcLC.begin(), srcLC.end(), srcLC.begin(), []( unsigned char c ) { return
static_cast<char>( std::tolower(c) ); } );
09086     if (srcLC == "y" || srcLC == "1" || srcLC == "true" || srcLC == "yes" || srcLC == "on")
09087         target = true;
09088     else if (srcLC == "n" || srcLC == "0" || srcLC == "false" || srcLC == "no" || srcLC == "off")
09089         target = false;
09090     else
09091         return ParserResult::runtimeError( "Expected a boolean value but did not recognise: '" +
source + "'" );
09092     return ParserResult::ok( ParseResultType::Matched );
09093 }
09094 #ifdef CLARA_CONFIG_OPTIONAL_TYPE
09095     template<typename T>
09096     inline auto convertInto( std::string const &source, CLARA_CONFIG_OPTIONAL_TYPE<T>& target ) ->
ParserResult {
09097         T temp;
09098         auto result = convertInto( source, temp );
09099         if( result )
09100             target = std::move(temp);
09101         return result;
09102     }
09103 #endif // CLARA_CONFIG_OPTIONAL_TYPE
09104
09105 struct NonCopyable {
09106     NonCopyable() = default;
09107     NonCopyable( NonCopyable const & ) = delete;
09108     NonCopyable( NonCopyable && ) = delete;
09109     NonCopyable &operator=( NonCopyable const & ) = delete;
09110     NonCopyable &operator=( NonCopyable && ) = delete;
09111 };
09112
09113 struct BoundRef : NonCopyable {
09114     virtual ~BoundRef() = default;
09115     virtual auto isContainer() const -> bool { return false; }
09116     virtual auto isFlag() const -> bool { return false; }
09117 };
09118 struct BoundValueRefBase : BoundRef {
09119     virtual auto setValue( std::string const &arg ) -> ParserResult = 0;
09120 };
09121 struct BoundFlagRefBase : BoundRef {
09122     virtual auto setFlag( bool flag ) -> ParserResult = 0;
09123     virtual auto isFlag() const -> bool { return true; }
09124 };
09125
09126 template<typename T>
09127 struct BoundValueRef : BoundValueRefBase {
09128     T &m_ref;

```

```

09129
09130     explicit BoundValueRef( T &ref ) : m_ref( ref ) {}
09131
09132     auto setValue( std::string const &arg ) -> ParserResult override {
09133         return convertInto( arg, m_ref );
09134     }
09135 };
09136
09137 template<typename T>
09138 struct BoundValueRef<std::vector<T> > : BoundValueRefBase {
09139     std::vector<T> &m_ref;
09140
09141     explicit BoundValueRef( std::vector<T> &ref ) : m_ref( ref ) {}
09142
09143     auto isContainer() const -> bool override { return true; }
09144
09145     auto setValue( std::string const &arg ) -> ParserResult override {
09146         T temp;
09147         auto result = convertInto( arg, temp );
09148         if( result )
09149             m_ref.push_back( temp );
09150         return result;
09151     }
09152 };
09153
09154 struct BoundFlagRef : BoundFlagRefBase {
09155     bool &m_ref;
09156
09157     explicit BoundFlagRef( bool &ref ) : m_ref( ref ) {}
09158
09159     auto setFlag( bool flag ) -> ParserResult override {
09160         m_ref = flag;
09161         return ParserResult::ok( ParseResultType::Matched );
09162     }
09163 };
09164
09165 template<typename Return Type>
09166 struct LambdaInvoker {
09167     static_assert( std::is_same<Return Type, ParserResult>::value, "Lambda must return void or
09168 clara::ParserResult" );
09169
09170     template<typename L, typename ArgType>
09171     static auto invoke( L const &lambda, ArgType const &arg ) -> ParserResult {
09172         return lambda( arg );
09173     }
09174 };
09175
09176 template<>
09177 struct LambdaInvoker<void> {
09178     template<typename L, typename ArgType>
09179     static auto invoke( L const &lambda, ArgType const &arg ) -> ParserResult {
09180         lambda( arg );
09181         return ParserResult::ok( ParseResultType::Matched );
09182     }
09183 };
09184
09185 template<typename ArgType, typename L>
09186 inline auto invokeLambda( L const &lambda, std::string const &arg ) -> ParserResult {
09187     ArgType temp{};
09188     auto result = convertInto( arg, temp );
09189     return !result
09190         ? result
09191         : LambdaInvoker<typename UnaryLambdaTraits<L>::Return Type>::invoke( lambda, temp );
09192 }
09193
09194 template<typename L>
09195 struct BoundLambda : BoundValueRefBase {
09196     L m_lambda;
09197
09198     static_assert( UnaryLambdaTraits<L>::isValid, "Supplied lambda must take exactly one argument"
09199 );
09200
09201     explicit BoundLambda( L const &lambda ) : m_lambda( lambda ) {}
09202
09203     auto setValue( std::string const &arg ) -> ParserResult override {
09204         return invokeLambda<typename UnaryLambdaTraits<L>::ArgType>( m_lambda, arg );
09205     }
09206 };
09207
09208 template<typename L>
09209 struct BoundFlagLambda : BoundFlagRefBase {
09210     L m_lambda;
09211
09212     static_assert( UnaryLambdaTraits<L>::isValid, "Supplied lambda must take exactly one argument"
09213 );
09214
09215     static_assert( std::is_same<typename UnaryLambdaTraits<L>::ArgType, bool>::value, "flags must
09216 be boolean" );
09217

```

```

09212     explicit BoundFlagLambda( L const &lambda ) : m_lambda( lambda ) {}
09213
09214     auto setFlag( bool flag ) -> ParserResult override {
09215         return LambdaInvoker<typename UnaryLambdaTraits<L>::ReturnType>::invoke( m_lambda, flag );
09216     }
09217 };
09218
09219 enum class Optionality { Optional, Required };
09220
09221 struct Parser;
09222
09223 class ParserBase {
09224 public:
09225     virtual ~ParserBase() = default;
09226     virtual auto validate() const -> Result { return Result::ok(); }
09227     virtual auto parse( std::string const& exeName, TokenStream const& tokens) const ->
InternalParseResult = 0;
09228     virtual auto cardinality() const -> size_t { return 1; }
09229
09230     auto parse( Args const& args ) const -> InternalParseResult {
09231         return parse( args.exeName(), TokenStream( args ) );
09232     }
09233 };
09234
09235 template<typename DerivedT>
09236 class ComposableParserImpl : public ParserBase {
09237 public:
09238     template<typename T>
09239     auto operator|( T const& other ) const -> Parser;
09240
09241     template<typename T>
09242     auto operator+( T const& other ) const -> Parser;
09243 };
09244
09245 // Common code and state for Args and Opts
09246 template<typename DerivedT>
09247 class ParserRefImpl : public ComposableParserImpl<DerivedT> {
09248 protected:
09249     Optionality m_optionality = Optionality::Optional;
09250     std::shared_ptr<BoundRef> m_ref;
09251     std::string m_hint;
09252     std::string m_description;
09253
09254     explicit ParserRefImpl( std::shared_ptr<BoundRef> const& ref ) : m_ref( ref ) {}
09255
09256 public:
09257     template<typename T>
09258     ParserRefImpl( T& ref, std::string const& hint )
09259     : m_ref( std::make_shared<BoundValueRef<T>( ref ) ),
09260       m_hint( hint )
09261     {}
09262
09263     template<typename LambdaT>
09264     ParserRefImpl( LambdaT const& ref, std::string const& hint )
09265     : m_ref( std::make_shared<BoundLambda<LambdaT>( ref ) ),
09266       m_hint( hint )
09267     {}
09268
09269     auto operator()( std::string const& description ) -> DerivedT & {
09270         m_description = description;
09271         return static_cast<DerivedT &>( *this );
09272     }
09273
09274     auto optional() -> DerivedT & {
09275         m_optionality = Optionality::Optional;
09276         return static_cast<DerivedT &>( *this );
09277     };
09278
09279     auto required() -> DerivedT & {
09280         m_optionality = Optionality::Required;
09281         return static_cast<DerivedT &>( *this );
09282     };
09283
09284     auto isOptional() const -> bool {
09285         return m_optionality == Optionality::Optional;
09286     }
09287
09288     auto cardinality() const -> size_t override {
09289         if( m_ref->isContainer() )
09290             return 0;
09291         else
09292             return 1;
09293     }
09294
09295     auto hint() const -> std::string { return m_hint; }
09296 };
09297

```

```

09298     class ExeName : public ComposableParserImpl<ExeName> {
09299         std::shared_ptr<std::string> m_name;
09300         std::shared_ptr<BoundValueRefBase> m_ref;
09301
09302         template<typename LambdaT>
09303         static auto makeRef(LambdaT const& lambda) -> std::shared_ptr<BoundValueRefBase> {
09304             return std::make_shared<BoundLambda<LambdaT>>( lambda );
09305         }
09306
09307     public:
09308         ExeName() : m_name( std::make_shared<std::string>( "<executable>" ) ) {}
09309
09310         explicit ExeName( std::string &ref ) : ExeName() {
09311             m_ref = std::make_shared<BoundValueRef<std::string>>( ref );
09312         }
09313
09314         template<typename LambdaT>
09315         explicit ExeName( LambdaT const& lambda ) : ExeName() {
09316             m_ref = std::make_shared<BoundLambda<LambdaT>>( lambda );
09317         }
09318
09319         // The exe name is not parsed out of the normal tokens, but is handled specially
09320         auto parse( std::string const&, TokenStream const& tokens ) const -> InternalParseResult
09321     override {
09322         return InternalParseResult::ok( ParseState( ParseResultType::NoMatch, tokens ) );
09323     }
09324
09325     auto name() const -> std::string { return *m_name; }
09326     auto set( std::string const& newName ) -> ParserResult {
09327
09328         auto lastSlash = newName.find_last_of( "\\/" );
09329         auto filename = ( lastSlash == std::string::npos )
09330             ? newName
09331             : newName.substr( lastSlash+1 );
09332
09333         *m_name = filename;
09334         if( m_ref )
09335             return m_ref->set_value( filename );
09336         else
09337             return ParserResult::ok( ParseResultType::Matched );
09338     }
09339 };
09340
09341     class Arg : public ParserRefImpl<Arg> {
09342     public:
09343         using ParserRefImpl::ParserRefImpl;
09344
09345         auto parse( std::string const&, TokenStream const& tokens ) const -> InternalParseResult
09346     override {
09347         auto validationResult = validate();
09348         if( !validationResult )
09349             return InternalParseResult( validationResult );
09350
09351         auto remainingTokens = tokens;
09352         auto const& token = *remainingTokens;
09353         if( token.type != TokenType::Argument )
09354             return InternalParseResult::ok( ParseState( ParseResultType::NoMatch, remainingTokens ) );
09355
09356         assert( !m_ref->isFlag() );
09357         auto valueRef = static_cast<detail::BoundValueRefBase*>( m_ref.get() );
09358
09359         auto result = valueRef->set_value( remainingTokens->token );
09360         if( !result )
09361             return InternalParseResult( result );
09362         else
09363             return InternalParseResult::ok( ParseState( ParseResultType::Matched,
09364                 ++remainingTokens ) );
09365     }
09366 };
09367
09368     inline auto normaliseOpt( std::string const& optName ) -> std::string {
09369 #ifndef CATCH_PLATFORM_WINDOWS
09370         if( optName[0] == '/' )
09371             return "-" + optName.substr( 1 );
09372         else
09373             return optName;
09374 #endif
09375     }
09376
09377     class Opt : public ParserRefImpl<Opt> {
09378     protected:
09379         std::vector<std::string> m_optNames;
09380
09381     public:
09382         template<typename LambdaT>
09383         explicit Opt( LambdaT const& ref ) : ParserRefImpl( std::make_shared<BoundFlagLambda<LambdaT>>(

```

```

    ref ) ) {}
09381
09382     explicit Opt( bool &ref ) : ParserRefImpl( std::make_shared<BoundFlagRef>( ref ) ) {}
09383
09384     template<typename LambdaT>
09385     Opt( LambdaT const &ref, std::string const &hint ) : ParserRefImpl( ref, hint ) {}
09386
09387     template<typename T>
09388     Opt( T &ref, std::string const &hint ) : ParserRefImpl( ref, hint ) {}
09389
09390     auto operator[]( std::string const &optName ) -> Opt & {
09391         m_optNames.push_back( optName );
09392         return *this;
09393     }
09394
09395     auto getHelpColumns() const -> std::vector<HelpColumns> {
09396         std::ostringstream oss;
09397         bool first = true;
09398         for( auto const &opt : m_optNames ) {
09399             if (first)
09400                 first = false;
09401             else
09402                 oss << ", ";
09403             oss << opt;
09404         }
09405         if( !m_hint.empty() )
09406             oss << " <" << m_hint << ">";
09407         return { { oss.str(), m_description } };
09408     }
09409
09410     auto isMatch( std::string const &optToken ) const -> bool {
09411         auto normalisedToken = normaliseOpt( optToken );
09412         for( auto const &name : m_optNames ) {
09413             if( normaliseOpt( name ) == normalisedToken )
09414                 return true;
09415         }
09416         return false;
09417     }
09418
09419     using ParserBase::parse;
09420
09421     auto parse( std::string const&, TokenStream const &tokens ) const -> InternalParseResult
09422     override {
09423         auto validationResult = validate();
09424         if( !validationResult )
09425             return InternalParseResult( validationResult );
09426
09427         auto remainingTokens = tokens;
09428         if( remainingTokens && remainingTokens->type == TokenType::Option ) {
09429             auto const &token = *remainingTokens;
09430             if( isMatch( token.token ) ) {
09431                 if( m_ref->isFlag() ) {
09432                     auto flagRef = static_cast<detail::BoundFlagRefBase*>( m_ref.get() );
09433                     auto result = flagRef->setFlag( true );
09434                     if( !result )
09435                         return InternalParseResult( result );
09436                     if( result.value() == ParseResultType::ShortCircuitAll )
09437                         return InternalParseResult::ok( ParseState( result.value(),
09438 remainingTokens ) );
09439                 } else {
09440                     auto valueRef = static_cast<detail::BoundValueRefBase*>( m_ref.get() );
09441                     ++remainingTokens;
09442                     if( !remainingTokens )
09443                         return InternalParseResult::runtimeError( "Expected argument following " +
09444 token.token );
09445                     auto const &argToken = *remainingTokens;
09446                     if( argToken.type != TokenType::Argument )
09447                         return InternalParseResult::runtimeError( "Expected argument following " +
09448 token.token );
09449                     auto result = valueRef->setValue( argToken.token );
09450                     if( !result )
09451                         return InternalParseResult( result );
09452                     if( result.value() == ParseResultType::ShortCircuitAll )
09453                         return InternalParseResult::ok( ParseState( result.value(),
09454 remainingTokens ) );
09455                 }
09456             }
09457             return InternalParseResult::ok( ParseState( ParseResultType::Matched,
09458 ++remainingTokens ) );
09459         }
09460         return InternalParseResult::ok( ParseState( ParseResultType::NoMatch, remainingTokens ) );
09461     }
09462
09463     auto validate() const -> Result override {
09464         if( m_optNames.empty() )
09465             return Result::logicError( "No options supplied to Opt" );
09466         for( auto const &name : m_optNames ) {

```



```

09461         if( name.empty() )
09462             return Result::logicError( "Option name cannot be empty" );
09463 #ifdef CATCH_PLATFORM_WINDOWS
09464         if( name[0] != '-' && name[0] != '/' )
09465             return Result::logicError( "Option name must begin with '-' or '/'" );
09466 #else
09467         if( name[0] != '-' )
09468             return Result::logicError( "Option name must begin with '-'" );
09469 #endif
09470     }
09471     return ParserRefImpl::validate();
09472 }
09473 };
09474
09475 struct Help : Opt {
09476     Help( bool &showHelpFlag )
09477     : Opt([&]( bool flag ) {
09478         showHelpFlag = flag;
09479         return ParserResult::ok( ParseResultType::ShortCircuitAll );
09480     })
09481     {
09482         static_cast<Opt &>( *this )
09483             ("display usage information")
09484             ["-?"]["-h"]["--help"]
09485             .optional();
09486     }
09487 };
09488
09489 struct Parser : ParserBase {
09490
09491     mutable ExeName m_exeName;
09492     std::vector<Opt> m_options;
09493     std::vector<Arg> m_args;
09494
09495     auto operator|=( ExeName const &exeName ) -> Parser & {
09496         m_exeName = exeName;
09497         return *this;
09498     }
09499
09500     auto operator|=( Arg const &arg ) -> Parser & {
09501         m_args.push_back(arg);
09502         return *this;
09503     }
09504
09505     auto operator|=( Opt const &opt ) -> Parser & {
09506         m_options.push_back(opt);
09507         return *this;
09508     }
09509
09510     auto operator|=( Parser const &other ) -> Parser & {
09511         m_options.insert(m_options.end(), other.m_options.begin(), other.m_options.end());
09512         m_args.insert(m_args.end(), other.m_args.begin(), other.m_args.end());
09513         return *this;
09514     }
09515
09516     template<typename T>
09517     auto operator|( T const &other ) const -> Parser {
09518         return Parser( *this ) |= other;
09519     }
09520
09521     // Forward deprecated interface with '+' instead of '|'
09522     template<typename T>
09523     auto operator+=( T const &other ) -> Parser & { return operator|=( other ); }
09524     template<typename T>
09525     auto operator+( T const &other ) const -> Parser { return operator|( other ); }
09526
09527     auto getHelpColumns() const -> std::vector<HelpColumns> {
09528         std::vector<HelpColumns> cols;
09529         for (auto const &o : m_options) {
09530             auto childCols = o.getHelpColumns();
09531             cols.insert( cols.end(), childCols.begin(), childCols.end() );
09532         }
09533         return cols;
09534     }
09535
09536     void writeToStream( std::ostream &os ) const {
09537         if (!m_exeName.name().empty()) {
09538             os << "usage:\n" << " " << m_exeName.name() << " ";
09539             bool required = true, first = true;
09540             for( auto const &arg : m_args ) {
09541                 if (first)
09542                     first = false;
09543                 else
09544                     os << " ";
09545                 if( arg.isOptional() && required ) {
09546                     os << "[";
09547                     required = false;

```

```

09548         }
09549         os << "<" << arg.hint() << ">";
09550         if( arg.cardinality() == 0 )
09551             os << " ... ";
09552     }
09553     if( !required )
09554         os << "]";
09555     if( !m_options.empty() )
09556         os << " options";
09557     os << "\n\nwhere options are:" << std::endl;
09558 }
09559
09560 auto rows = getHelpColumns();
09561 size_t consoleWidth = CATCH_CLARA_CONFIG_CONSOLE_WIDTH;
09562 size_t optWidth = 0;
09563 for( auto const &cols : rows )
09564     optWidth = (std::max)(optWidth, cols.left.size() + 2);
09565
09566 optWidth = (std::min)(optWidth, consoleWidth/2);
09567
09568 for( auto const &cols : rows ) {
09569     auto row =
09570         TextFlow::Column( cols.left ).width( optWidth ).indent( 2 ) +
09571         TextFlow::Spacer(4) +
09572         TextFlow::Column( cols.right ).width( consoleWidth - 7 - optWidth );
09573     os << row << std::endl;
09574 }
09575 }
09576
09577 friend auto operator<<( std::ostream &os, Parser const &parser ) -> std::ostream& {
09578     parser.writeToStream( os );
09579     return os;
09580 }
09581
09582 auto validate() const -> Result override {
09583     for( auto const &opt : m_options ) {
09584         auto result = opt.validate();
09585         if( !result )
09586             return result;
09587     }
09588     for( auto const &arg : m_args ) {
09589         auto result = arg.validate();
09590         if( !result )
09591             return result;
09592     }
09593     return Result::ok();
09594 }
09595
09596 using ParserBase::parse;
09597
09598 auto parse( std::string const& exeName, TokenStream const &tokens ) const ->
InternalParseResult override {
09599
09600     struct ParserInfo {
09601         ParserBase const* parser = nullptr;
09602         size_t count = 0;
09603     };
09604     const size_t totalParsers = m_options.size() + m_args.size();
09605     assert( totalParsers < 512 );
09606     // ParserInfo parseInfos[totalParsers]; // <-- this is what we really want to do
09607     ParserInfo parseInfos[512];
09608
09609     {
09610         size_t i = 0;
09611         for (auto const &opt : m_options) parseInfos[i++].parser = &opt;
09612         for (auto const &arg : m_args) parseInfos[i++].parser = &arg;
09613     }
09614
09615     m_exeName.set( exeName );
09616
09617     auto result = InternalParseResult::ok( ParseState( ParseResultType::NoMatch, tokens ) );
09618     while( result.value().remainingTokens() ) {
09619         bool tokenParsed = false;
09620
09621         for( size_t i = 0; i < totalParsers; ++i ) {
09622             auto& parseInfo = parseInfos[i];
09623             if( parseInfo.parser->cardinality() == 0 || parseInfo.count <
parseInfo.parser->cardinality() ) {
09624                 result = parseInfo.parser->parse(exeName, result.value().remainingTokens());
09625                 if (!result)
09626                     return result;
09627                 if (result.value().type() != ParseResultType::NoMatch) {
09628                     tokenParsed = true;
09629                     ++parseInfo.count;
09630                     break;
09631                 }
09632             }

```

```

09633     }
09634
09635     if( result.value().type() == ParseResultType::ShortCircuitAll )
09636         return result;
09637     if( !tokenParsed )
09638         return InternalParseResult::runtimeError( "Unrecognised token: " +
result.value().remainingTokens()->token );
09639     }
09640     // !TBD Check missing required options
09641     return result;
09642 }
09643 };
09644
09645 template<typename DerivedT>
09646 template<typename T>
09647 auto ComposableParserImpl<DerivedT>::operator|( T const& other ) const -> Parser {
09648     return Parser() | static_cast<DerivedT const&>( *this ) | other;
09649 }
09650 } // namespace detail
09651
09652 // A Combined parser
09653 using detail::Parser;
09654
09655 // A parser for options
09656 using detail::Opt;
09657
09658 // A parser for arguments
09659 using detail::Arg;
09660
09661 // Wrapper for argc, argv from main()
09662 using detail::Args;
09663
09664 // Specifies the name of the executable
09665 using detail::ExeName;
09666
09667 // Convenience wrapper for option parser that specifies the help option
09668 using detail::Help;
09669
09670 // enum of result types from a parse
09671 using detail::ParseResultType;
09672
09673 // Result type for parser operation
09674 using detail::ParserResult;
09675
09676 } // namespace Catch::clara
09677
09678 // end clara.hpp
09679 #ifdef __clang__
09680 #pragma clang diagnostic pop
09681 #endif
09682
09683 // Restore Clara's value for console width, if present
09684 #ifdef CATCH_TEMP_CLARA_CONFIG_CONSOLE_WIDTH
09685 #define CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH CATCH_TEMP_CLARA_CONFIG_CONSOLE_WIDTH
09686 #undef CATCH_TEMP_CLARA_CONFIG_CONSOLE_WIDTH
09687 #endif
09688
09689 // end catch_clara.h
09690 namespace Catch {
09691
09692     clara::Parser makeCommandLineParser( ConfigData& config );
09693
09694 } // end namespace Catch
09695
09696 // end catch_commandline.h
09697 #include <fstream>
09698 #include <ctime>
09699
09700 namespace Catch {
09701
09702     clara::Parser makeCommandLineParser( ConfigData& config ) {
09703
09704         using namespace clara;
09705
09706         auto const setWarning = [&]( std::string const& warning ) {
09707             auto warningSet = [&]() {
09708                 if( warning == "NoAssertions" )
09709                     return WarnAbout::NoAssertions;
09710
09711                 if ( warning == "NoTests" )
09712                     return WarnAbout::NoTests;
09713
09714                 return WarnAbout::Nothing;
09715             }();
09716
09717             if (warningSet == WarnAbout::Nothing)
09718                 return ParserResult::runtimeError( "Unrecognised warning: '" + warning + "'" );

```

```

09719         config.warnings = static_cast<WarnAbout::What>( config.warnings | warningSet );
09720         return ParserResult::ok( ParseResultType::Matched );
09721     };
09722     auto const loadTestNamesFromFile = [&]( std::string const& filename ) {
09723         std::ifstream f( filename.c_str() );
09724         if( !f.is_open() )
09725             return ParserResult::runtimeError( "Unable to load input file: '" + filename + "'"
09726 );
09727         std::string line;
09728         while( std::getline( f, line ) ) {
09729             line = trim(line);
09730             if( !line.empty() && !startsWith( line, '#' ) ) {
09731                 if( !startsWith( line, '"' ) )
09732                     line = '"' + line + '"';
09733                 config.testsOrTags.push_back( line );
09734                 config.testsOrTags.emplace_back( "," );
09735             }
09736         }
09737         //Remove comma in the end
09738         if(!config.testsOrTags.empty())
09739             config.testsOrTags.erase( config.testsOrTags.end()-1 );
09740
09741         return ParserResult::ok( ParseResultType::Matched );
09742     };
09743     auto const setTestOrder = [&]( std::string const& order ) {
09744         if( startsWith( "declared", order ) )
09745             config.runOrder = RunTests::InDeclarationOrder;
09746         else if( startsWith( "lexical", order ) )
09747             config.runOrder = RunTests::InLexicographicalOrder;
09748         else if( startsWith( "random", order ) )
09749             config.runOrder = RunTests::InRandomOrder;
09750         else
09751             return clara::ParserResult::runtimeError( "Unrecognised ordering: '" + order + "'"
09752 );
09753         return ParserResult::ok( ParseResultType::Matched );
09754     };
09755     auto const setRngSeed = [&]( std::string const& seed ) {
09756         if( seed != "time" )
09757             return clara::detail::convertInto( seed, config.rngSeed );
09758         config.rngSeed = static_cast<unsigned int>( std::time(nullptr) );
09759         return ParserResult::ok( ParseResultType::Matched );
09760     };
09761     auto const setColourUsage = [&]( std::string const& useColour ) {
09762         auto mode = toLower( useColour );
09763         if( mode == "yes" )
09764             config.useColour = UseColour::Yes;
09765         else if( mode == "no" )
09766             config.useColour = UseColour::No;
09767         else if( mode == "auto" )
09768             config.useColour = UseColour::Auto;
09769         else
09770             return ParserResult::runtimeError( "colour mode must be one of: auto, yes or
09771 no. '" + useColour + "' not recognised" );
09772         return ParserResult::ok( ParseResultType::Matched );
09773     };
09774     auto const setWaitForKeypress = [&]( std::string const& keypress ) {
09775         auto keypressLc = toLower( keypress );
09776         if( keypressLc == "never" )
09777             config.waitForKeypress = WaitForKeypress::Never;
09778         else if( keypressLc == "start" )
09779             config.waitForKeypress = WaitForKeypress::BeforeStart;
09780         else if( keypressLc == "exit" )
09781             config.waitForKeypress = WaitForKeypress::BeforeExit;
09782         else if( keypressLc == "both" )
09783             config.waitForKeypress = WaitForKeypress::BeforeStartAndExit;
09784         else
09785             return ParserResult::runtimeError( "keypress argument must be one of: never,
09786 start, exit or both. '" + keypress + "' not recognised" );
09787         return ParserResult::ok( ParseResultType::Matched );
09788     };
09789     auto const setVerbosity = [&]( std::string const& verbosity ) {
09790         auto lcVerbosity = toLower( verbosity );
09791         if( lcVerbosity == "quiet" )
09792             config.verbosity = Verbosity::Quiet;
09793         else if( lcVerbosity == "normal" )
09794             config.verbosity = Verbosity::Normal;
09795         else if( lcVerbosity == "high" )
09796             config.verbosity = Verbosity::High;
09797         else
09798             return ParserResult::runtimeError( "Unrecognised verbosity, '" + verbosity + "'" );
09799         return ParserResult::ok( ParseResultType::Matched );
09800     };
09801     auto const setReporter = [&]( std::string const& reporter ) {
09802         IReporterRegistry::FactoryMap const& factories =
09803             getRegistryHub().getReporterRegistry().getFactories();

```

```

09801
09802     auto lcReporter = toLower( reporter );
09803     auto result = factories.find( lcReporter );
09804
09805     if( factories.end() != result )
09806         config.reporterName = lcReporter;
09807     else
09808         return ParserResult::runtimeError( "Unrecognized reporter, '" + reporter + "'. Check
available with --list-reporters" );
09809     return ParserResult::ok( ParseResultType::Matched );
09810 };
09811
09812 auto cli
09813 = ExeName( config.processName )
09814 | Help( config.showHelp )
09815 | Opt( config.listTests )
09816   [ "-l" ] [ "--list-tests" ]
09817   ( "list all/matching test cases" )
09818 | Opt( config.listTags )
09819   [ "-t" ] [ "--list-tags" ]
09820   ( "list all/matching tags" )
09821 | Opt( config.showSuccessfulTests )
09822   [ "-s" ] [ "--success" ]
09823   ( "include successful tests in output" )
09824 | Opt( config.shouldDebugBreak )
09825   [ "-b" ] [ "--break" ]
09826   ( "break into debugger on failure" )
09827 | Opt( config.noThrow )
09828   [ "-e" ] [ "--nothrow" ]
09829   ( "skip exception tests" )
09830 | Opt( config.showInvisibles )
09831   [ "-i" ] [ "--invisibles" ]
09832   ( "show invisibles (tabs, newlines)" )
09833 | Opt( config.outputFilename, "filename" )
09834   [ "-o" ] [ "--out" ]
09835   ( "output filename" )
09836 | Opt( setReporter, "name" )
09837   [ "-r" ] [ "--reporter" ]
09838   ( "reporter to use (defaults to console)" )
09839 | Opt( config.name, "name" )
09840   [ "-n" ] [ "--name" ]
09841   ( "suite name" )
09842 | Opt( [&]( bool ){ config.abortAfter = 1; } )
09843   [ "-a" ] [ "--abort" ]
09844   ( "abort at first failure" )
09845 | Opt( [&]( int x ){ config.abortAfter = x; }, "no. failures" )
09846   [ "-x" ] [ "--abortx" ]
09847   ( "abort after x failures" )
09848 | Opt( setWarning, "warning name" )
09849   [ "-w" ] [ "--warn" ]
09850   ( "enable warnings" )
09851 | Opt( [&]( bool flag ) { config.showDurations = flag ? ShowDurations::Always :
ShowDurations::Never; }, "yes|no" )
09852   [ "-d" ] [ "--durations" ]
09853   ( "show test durations" )
09854 | Opt( config.minDuration, "seconds" )
09855   [ "-D" ] [ "--min-duration" ]
09856   ( "show test durations for tests taking at least the given number of seconds" )
09857 | Opt( loadTestNamesFromFile, "filename" )
09858   [ "-f" ] [ "--input-file" ]
09859   ( "load test names to run from a file" )
09860 | Opt( config.fileNamesAsTags )
09861   [ "-#" ] [ "--filenames-as-tags" ]
09862   ( "adds a tag for the filename" )
09863 | Opt( config.sectionsToRun, "section name" )
09864   [ "-c" ] [ "--section" ]
09865   ( "specify section to run" )
09866 | Opt( setVerbosity, "quiet|normal|high" )
09867   [ "-v" ] [ "--verbosity" ]
09868   ( "set output verbosity" )
09869 | Opt( config.listTestNamesOnly )
09870   [ "--list-test-names-only" ]
09871   ( "list all/matching test cases names only" )
09872 | Opt( config.listReporters )
09873   [ "--list-reporters" ]
09874   ( "list all reporters" )
09875 | Opt( setTestOrder, "decl|lex|rand" )
09876   [ "--order" ]
09877   ( "test case order (defaults to decl)" )
09878 | Opt( setRngSeed, "'time'|number" )
09879   [ "--rng-seed" ]
09880   ( "set a specific seed for random numbers" )
09881 | Opt( setColourUsage, "yes|no" )
09882   [ "--use-colour" ]
09883   ( "should output be colourised" )
09884 | Opt( config.libIdentify )
09885   [ "--libidentify" ]

```

```

09886         ( "report name and version according to libidentify standard" )
09887     | Opt( setWaitForKeypress, "never|start|exit|both" )
09888         [ "--wait-for-keypress" ]
09889         ( "waits for a keypress before exiting" )
09890     | Opt( config.benchmarkSamples, "samples" )
09891         [ "--benchmark-samples" ]
09892         ( "number of samples to collect (default: 100)" )
09893     | Opt( config.benchmarkResamples, "resamples" )
09894         [ "--benchmark-resamples" ]
09895         ( "number of resamples for the bootstrap (default: 100000)" )
09896     | Opt( config.benchmarkConfidenceInterval, "confidence interval" )
09897         [ "--benchmark-confidence-interval" ]
09898         ( "confidence interval for the bootstrap (between 0 and 1, default: 0.95)" )
09899     | Opt( config.benchmarkNoAnalysis, "no-analysis" )
09900         [ "--benchmark-no-analysis" ]
09901         ( "perform only measurements; do not perform any analysis" )
09902     | Opt( config.benchmarkWarmupTime, "benchmarkWarmupTime" )
09903         [ "--benchmark-warmup-time" ]
09904         ( "amount of time in milliseconds spent on warming up each test (default: 100)" )
09905     | Arg( config.testsOrTags, "test name|pattern|tags" )
09906         ( "which test or tests to use" );
09907
09908     return cli;
09909 }
09910
09911 } // end namespace Catch
09912 // end catch_commandline.cpp
09913 // start catch_common.cpp
09914
09915 #include <cstring>
09916 #include <ostream>
09917
09918 namespace Catch {
09919
09920     bool SourceLineInfo::operator == ( SourceLineInfo const& other ) const noexcept {
09921         return line == other.line && (file == other.file || std::strcmp(file, other.file) == 0);
09922     }
09923
09924     bool SourceLineInfo::operator < ( SourceLineInfo const& other ) const noexcept {
09925         // We can assume that the same file will usually have the same pointer.
09926         // Thus, if the pointers are the same, there is no point in calling the strcmp
09927         return line < other.line || ( line == other.line && file != other.file && (std::strcmp(file,
09928         other.file) < 0));
09929     }
09930
09931     std::ostream& operator << ( std::ostream& os, SourceLineInfo const& info ) {
09932 #ifndef __GNUG__
09933         os << info.file << '(' << info.line << ')';
09934 #else
09935         os << info.file << ':' << info.line;
09936 #endif
09937         return os;
09938     }
09939
09940     std::string StreamEndStop::operator+() const {
09941         return std::string();
09942     }
09943
09944     NonCopyable::NonCopyable() = default;
09945     NonCopyable::~NonCopyable() = default;
09946 }
09947 // end catch_common.cpp
09948 // start catch_config.cpp
09949
09950 namespace Catch {
09951
09952     Config::Config( ConfigData const& data )
09953     :   m_data( data ),
09954         m_stream( openStream() )
09955     {
09956         // We need to trim filter specs to avoid trouble with superfluous
09957         // whitespace (esp. important for bdd macros, as those are manually
09958         // aligned with whitespace).
09959         for (auto& elem : m_data.testsOrTags) {
09960             elem = trim(elem);
09961         }
09962         for (auto& elem : m_data.sectionsToRun) {
09963             elem = trim(elem);
09964         }
09965
09966         TestSpecParser parser(ITagAliasRegistry::get());
09967         if (!m_data.testsOrTags.empty()) {
09968             m_hasTestFilters = true;
09969             for (auto const& testOrTags : m_data.testsOrTags) {
09970                 parser.parse(testOrTags);
09971             }
09972         }
09973     }

```

```

09972     }
09973     m_testSpec = parser.testSpec();
09974 }
09975
09976 std::string const& Config::getFilename() const {
09977     return m_data.outputFilename;
09978 }
09979
09980 bool Config::listTests() const { return m_data.listTests; }
09981 bool Config::listTestNamesOnly() const { return m_data.listTestNamesOnly; }
09982 bool Config::listTags() const { return m_data.listTags; }
09983 bool Config::listReporters() const { return m_data.listReporters; }
09984
09985 std::string Config::getProcessName() const { return m_data.processName; }
09986 std::string const& Config::getReporterName() const { return m_data.reporterName; }
09987
09988 std::vector<std::string> const& Config::getTestsOrTags() const { return m_data.testsOrTags; }
09989 std::vector<std::string> const& Config::getSectionsToRun() const { return m_data.sectionsToRun; }
09990
09991 TestSpec const& Config::testSpec() const { return m_testSpec; }
09992 bool Config::hasTestFilters() const { return m_hasTestFilters; }
09993
09994 bool Config::showHelp() const { return m_data.showHelp; }
09995
09996 // IConfig interface
09997 bool Config::allowThrows() const { return !m_data.noThrow; }
09998 std::ostream& Config::stream() const { return m_stream->stream(); }
09999 std::string Config::name() const { return m_data.name.empty() ?
m_data.processName : m_data.name; }
10000 bool Config::includeSuccessfulResults() const { return m_data.showSuccessfulTests; }
10001 bool Config::warnAboutMissingAssertions() const { return !(m_data.warnings &
WarnAbout::NoAssertions); }
10002 bool Config::warnAboutNoTests() const { return !(m_data.warnings &
WarnAbout::NoTests); }
10003 ShowDurations::OrNot Config::showDurations() const { return m_data.showDurations; }
10004 double Config::minDuration() const { return m_data.minDuration; }
10005 RunTests::InWhatOrder Config::runOrder() const { return m_data.runOrder; }
10006 unsigned int Config::rngSeed() const { return m_data.rngSeed; }
10007 UseColour::YesOrNo Config::useColour() const { return m_data.useColour; }
10008 bool Config::shouldDebugBreak() const { return m_data.shouldDebugBreak; }
10009 int Config::abortAfter() const { return m_data.abortAfter; }
10010 bool Config::showInvisibles() const { return m_data.showInvisibles; }
10011 Verbosity Config::verbosity() const { return m_data.verbosity; }
10012
10013 bool Config::benchmarkNoAnalysis() const { return m_data.benchmarkNoAnalysis; }
10014 int Config::benchmarkSamples() const { return m_data.benchmarkSamples; }
10015 double Config::benchmarkConfidenceInterval() const { return
m_data.benchmarkConfidenceInterval; }
10016 unsigned int Config::benchmarkResamples() const { return m_data.benchmarkResamples; }
10017
10018 std::chrono::milliseconds Config::benchmarkWarmupTime() const { return
std::chrono::milliseconds(m_data.benchmarkWarmupTime); }
10019
10019 IStream const* Config::openStream() {
10020     return Catch::makeStream(m_data.outputFilename);
10021 }
10022
10023 } // end namespace Catch
10024 // end catch_config.cpp
10025 // start catch_console_colour.cpp
10026
10027 #if defined(__clang__)
10028 #    pragma clang diagnostic push
10029 #    pragma clang diagnostic ignored "-Wexit-time-destructors"
10030 #endif
10031
10032 // start catch_errno_guard.h
10033
10034 namespace Catch {
10035
10036     class ErrnoGuard {
10037     public:
10038         ErrnoGuard();
10039         ~ErrnoGuard();
10040     private:
10041         int m_oldErrno;
10042     };
10043
10044 }
10045
10046 // end catch_errno_guard.h
10047 // start catch_windows_h_proxy.h
10048
10049
10050 #if defined(CATCH_PLATFORM_WINDOWS)
10051

```

```

10052 #if !defined(NOMINMAX) && !defined(CATCH_CONFIG_NO_NOMINMAX)
10053 #   define CATCH_DEFINED_NOMINMAX
10054 #   define NOMINMAX
10055 #endif
10056 #if !defined(WIN32_LEAN_AND_MEAN) && !defined(CATCH_CONFIG_NO_WIN32_LEAN_AND_MEAN)
10057 #   define CATCH_DEFINED_WIN32_LEAN_AND_MEAN
10058 #   define WIN32_LEAN_AND_MEAN
10059 #endif
10060
10061 #ifdef __AFXDLL
10062 #include <afxWin.h>
10063 #else
10064 #include <windows.h>
10065 #endif
10066
10067 #ifdef CATCH_DEFINED_NOMINMAX
10068 #   undef NOMINMAX
10069 #endif
10070 #ifdef CATCH_DEFINED_WIN32_LEAN_AND_MEAN
10071 #   undef WIN32_LEAN_AND_MEAN
10072 #endif
10073
10074 #endif // defined(CATCH_PLATFORM_WINDOWS)
10075
10076 // end catch_windows_h_proxy.h
10077 #include <sstream>
10078
10079 namespace Catch {
10080     namespace {
10081
10082         struct IColourImpl {
10083             virtual ~IColourImpl() = default;
10084             virtual void use( Colour::Code _colourCode ) = 0;
10085         };
10086
10087         struct NoColourImpl : IColourImpl {
10088             void use( Colour::Code ) override {}
10089
10090             static IColourImpl* instance() {
10091                 static NoColourImpl s_instance;
10092                 return &s_instance;
10093             }
10094         };
10095     } // anon namespace
10096 } // namespace Catch
10097
10098 #if !defined( CATCH_CONFIG_COLOUR_NONE ) && !defined( CATCH_CONFIG_COLOUR_WINDOWS ) && !defined(
10099     CATCH_CONFIG_COLOUR_ANSI )
10100 #   ifdef CATCH_PLATFORM_WINDOWS
10101 #       define CATCH_CONFIG_COLOUR_WINDOWS
10102 #   else
10103 #       define CATCH_CONFIG_COLOUR_ANSI
10104 #   endif
10105 #endif
10106
10107 #if defined ( CATCH_CONFIG_COLOUR_WINDOWS )
10108
10109 namespace Catch {
10110     namespace {
10111
10112         class Win32ColourImpl : public IColourImpl {
10113         public:
10114             Win32ColourImpl() : stdoutHandle( GetStdHandle(STD_OUTPUT_HANDLE) )
10115             {
10116                 CONSOLE_SCREEN_BUFFER_INFO csbiInfo;
10117                 GetConsoleScreenBufferInfo( stdoutHandle, &csbiInfo );
10118                 originalForegroundAttributes = csbiInfo.wAttributes & ~( BACKGROUND_GREEN | BACKGROUND_RED
10119 | BACKGROUND_BLUE | BACKGROUND_INTENSITY );
10119                 originalBackgroundAttributes = csbiInfo.wAttributes & ~( FOREGROUND_GREEN | FOREGROUND_RED
10120 | FOREGROUND_BLUE | FOREGROUND_INTENSITY );
10121             }
10122
10123             void use( Colour::Code _colourCode ) override {
10124                 switch( _colourCode ) {
10125                     case Colour::None:           return setTextAttribute( originalForegroundAttributes );
10126                     case Colour::White:          return setTextAttribute( FOREGROUND_GREEN | FOREGROUND_RED |
10127 FOREGROUND_BLUE );
10128                     case Colour::Red:            return setTextAttribute( FOREGROUND_RED );
10129                     case Colour::Green:          return setTextAttribute( FOREGROUND_GREEN );
10130                     case Colour::Blue:          return setTextAttribute( FOREGROUND_BLUE );
10131                     case Colour::Cyan:          return setTextAttribute( FOREGROUND_BLUE | FOREGROUND_GREEN );
10132                     case Colour::Yellow:         return setTextAttribute( FOREGROUND_RED | FOREGROUND_GREEN );
10133                     case Colour::Grey:          return setTextAttribute( 0 );
10134
10135                     case Colour::LightGrey:      return setTextAttribute( FOREGROUND_INTENSITY );
10136                     case Colour::BrightRed:     return setTextAttribute( FOREGROUND_INTENSITY |

```



```

    FOREGROUND_RED );
10135         case Colour::BrightGreen: return setTextAttribute( FOREGROUND_INTENSITY |
    FOREGROUND_GREEN );
10136         case Colour::BrightWhite: return setTextAttribute( FOREGROUND_INTENSITY |
    FOREGROUND_GREEN | FOREGROUND_RED | FOREGROUND_BLUE );
10137         case Colour::BrightYellow: return setTextAttribute( FOREGROUND_INTENSITY |
    FOREGROUND_RED | FOREGROUND_GREEN );
10138
10139         case Colour::Bright: CATCH_INTERNAL_ERROR( "not a colour" );
10140
10141         default:
10142             CATCH_ERROR( "Unknown colour requested" );
10143     }
10144 }
10145
10146 private:
10147     void setTextAttribute( WORD _textAttribute ) {
10148         SetConsoleTextAttribute( stdoutHandle, _textAttribute | originalBackgroundAttributes );
10149     }
10150     HANDLE stdoutHandle;
10151     WORD originalForegroundAttributes;
10152     WORD originalBackgroundAttributes;
10153 };
10154
10155 IColourImpl* platformColourInstance() {
10156     static Win32ColourImpl s_instance;
10157
10158     IConfigPtr config = getCurrentContext().getConfig();
10159     UseColour::YesOrNo colourMode = config
10160         ? config->useColour()
10161         : UseColour::Auto;
10162     if( colourMode == UseColour::Auto )
10163         colourMode = UseColour::Yes;
10164     return colourMode == UseColour::Yes
10165         ? &s_instance
10166         : NoColourImpl::instance();
10167 }
10168
10169 } // end anon namespace
10170 } // end namespace Catch
10171
10172 #elif defined( CATCH_CONFIG_COLOUR_ANSI )
10173
10174 #include <unistd.h>
10175
10176 namespace Catch {
10177 namespace {
10178
10179     // use POSIX/ ANSI console terminal codes
10180     // Thanks to Adam Strzelecki for original contribution
10181     // (http://github.com/nanoant)
10182     // https://github.com/philsquared/Catch/pull/131
10183     class PosixColourImpl : public IColourImpl {
10184     public:
10185         void use( Colour::Code _colourCode ) override {
10186             switch( _colourCode ) {
10187                 case Colour::None:
10188                 case Colour::White: return setColour( "[0m" );
10189                 case Colour::Red: return setColour( "[0;31m" );
10190                 case Colour::Green: return setColour( "[0;32m" );
10191                 case Colour::Blue: return setColour( "[0;34m" );
10192                 case Colour::Cyan: return setColour( "[0;36m" );
10193                 case Colour::Yellow: return setColour( "[0;33m" );
10194                 case Colour::Grey: return setColour( "[1;30m" );
10195
10196                 case Colour::LightGrey: return setColour( "[0;37m" );
10197                 case Colour::BrightRed: return setColour( "[1;31m" );
10198                 case Colour::BrightGreen: return setColour( "[1;32m" );
10199                 case Colour::BrightWhite: return setColour( "[1;37m" );
10200                 case Colour::BrightYellow: return setColour( "[1;33m" );
10201
10202                 case Colour::Bright: CATCH_INTERNAL_ERROR( "not a colour" );
10203                 default: CATCH_INTERNAL_ERROR( "Unknown colour requested" );
10204             }
10205         }
10206         static IColourImpl* instance() {
10207             static PosixColourImpl s_instance;
10208             return &s_instance;
10209         }
10210
10211     private:
10212         void setColour( const char* _escapeCode ) {
10213             getCurrentContext().getConfig()->stream()
10214                 << "\033" << _escapeCode;
10215         }
10216     };
10217 }

```

```

10218     bool useColourOnPlatform() {
10219         return
10220 #if defined(CATCH_PLATFORM_MAC) || defined(CATCH_PLATFORM_IPHONE)
10221         !isDebuggerActive() &&
10222 #endif
10223 #if !(defined(__DJGPP__) && defined(__STRICT_ANSI__))
10224         isatty(STDOUT_FILENO)
10225 #else
10226         false
10227 #endif
10228     };
10229 }
10230 IColourImpl* platformColourInstance() {
10231     ErrnoGuard guard;
10232     IConfigPtr config = getCurrentContext().getConfig();
10233     UseColour::YesOrNo colourMode = config
10234     ? config->useColour()
10235     : UseColour::Auto;
10236     if( colourMode == UseColour::Auto )
10237         colourMode = useColourOnPlatform()
10238         ? UseColour::Yes
10239         : UseColour::No;
10240     return colourMode == UseColour::Yes
10241         ? PosixColourImpl::instance()
10242         : NoColourImpl::instance();
10243 }
10244
10245 } // end anon namespace
10246 } // end namespace Catch
10247
10248 #else // not Windows or ANSI //////////////////////////////////////
10249 namespace Catch {
10250
10251     static IColourImpl* platformColourInstance() { return NoColourImpl::instance(); }
10252
10253 } // end namespace Catch
10254
10255 #endif // Windows/ ANSI/ None
10256
10257 namespace Catch {
10258
10259     Colour::Colour( Code _colourCode ) { use( _colourCode ); }
10260     Colour::Colour( Colour&& other ) noexcept {
10261         m_moved = other.m_moved;
10262         other.m_moved = true;
10263     }
10264     Colour& Colour::operator=( Colour&& other ) noexcept {
10265         m_moved = other.m_moved;
10266         other.m_moved = true;
10267         return *this;
10268     }
10269 }
10270
10271 Colour::~~Colour(){ if( !m_moved ) use( None ); }
10272
10273 void Colour::use( Code _colourCode ) {
10274     static IColourImpl* impl = platformColourInstance();
10275     // Strictly speaking, this cannot possibly happen.
10276     // However, under some conditions it does happen (see #1626),
10277     // and this change is small enough that we can let practicality
10278     // triumph over purity in this case.
10279     if (impl != nullptr) {
10280         impl->use( _colourCode );
10281     }
10282 }
10283
10284 std::ostream& operator << ( std::ostream& os, Colour const& ) {
10285     return os;
10286 }
10287
10288 } // end namespace Catch
10289
10290 #if defined(__clang__)
10291 # pragma clang diagnostic pop
10292 #endif
10293
10294 // end catch_console_colour.cpp
10295 // start catch_context.cpp
10296
10297 namespace Catch {
10298
10299     class Context : public IMutableContext, NonCopyable {
10300     public: // IContext
10301         IResultCapture* getResultCapture() override {
10302             return m_resultCapture;
10303         }
10304     }

```

```

10305     IRunner* getRunner() override {
10306         return m_runner;
10307     }
10308
10309     IConfigPtr const& getConfig() const override {
10310         return m_config;
10311     }
10312
10313     ~Context() override;
10314
10315     public: // IMutableContext
10316         void setResultCapture( IResultCapture* resultCapture ) override {
10317             m_resultCapture = resultCapture;
10318         }
10319         void setRunner( IRunner* runner ) override {
10320             m_runner = runner;
10321         }
10322         void setConfig( IConfigPtr const& config ) override {
10323             m_config = config;
10324         }
10325
10326         friend IMutableContext& getCurrentMutableContext();
10327
10328     private:
10329         IConfigPtr m_config;
10330         IRunner* m_runner = nullptr;
10331         IResultCapture* m_resultCapture = nullptr;
10332     };
10333
10334     IMutableContext *IMutableContext::currentContext = nullptr;
10335
10336     void IMutableContext::createContext()
10337     {
10338         currentContext = new Context();
10339     }
10340
10341     void cleanUpContext() {
10342         delete IMutableContext::currentContext;
10343         IMutableContext::currentContext = nullptr;
10344     }
10345     IContext::~IContext() = default;
10346     IMutableContext::~IMutableContext() = default;
10347     Context::~~Context() = default;
10348
10349     SimplePcg32& rng() {
10350         static SimplePcg32 s_rng;
10351         return s_rng;
10352     }
10353
10354 }
10355 // end catch_context.cpp
10356 // start catch_debug_console.cpp
10357
10358 // start catch_debug_console.h
10359
10360 #include <string>
10361
10362 namespace Catch {
10363     void writeToDebugConsole( std::string const& text );
10364 }
10365
10366 // end catch_debug_console.h
10367 #if defined(CATCH_CONFIG_ANDROID_LOGWRITE)
10368 #include <android/log.h>
10369
10370 namespace Catch {
10371     void writeToDebugConsole( std::string const& text ) {
10372         __android_log_write( ANDROID_LOG_DEBUG, "Catch", text.c_str() );
10373     }
10374 }
10375
10376 #elif defined(CATCH_PLATFORM_WINDOWS)
10377
10378 namespace Catch {
10379     void writeToDebugConsole( std::string const& text ) {
10380         ::OutputDebugStringA( text.c_str() );
10381     }
10382 }
10383
10384 #else
10385
10386 namespace Catch {
10387     void writeToDebugConsole( std::string const& text ) {
10388         // !TBD: Need a version for Mac/ XCode and other IDEs
10389         Catch::cout() << text;
10390     }
10391 }

```

```

10392
10393 #endif // Platform
10394 // end catch_debug_console.cpp
10395 // start catch_debugger.cpp
10396
10397 #if defined(CATCH_PLATFORM_MAC) || defined(CATCH_PLATFORM_IPHONE)
10398
10399 # include <cassert>
10400 # include <sys/types.h>
10401 # include <unistd.h>
10402 # include <cstdint>
10403 # include <ostream>
10404
10405 #ifdef __apple_build_version__
10406     // These headers will only compile with AppleClang (XCode)
10407     // For other compilers (Clang, GCC, ... ) we need to exclude them
10408 # include <sys/sysctl.h>
10409 #endif
10410
10411 namespace Catch {
10412     #ifdef __apple_build_version__
10413         // The following function is taken directly from the following technical note:
10414         // https://developer.apple.com/library/archive/qa/qa1361/_index.html
10415
10416         // Returns true if the current process is being debugged (either
10417         // running under the debugger or has a debugger attached post facto).
10418         bool isDebuggerActive() {
10419             int mib[4];
10420             struct kinfo_proc info;
10421             std::size_t size;
10422
10423             // Initialize the flags so that, if sysctl fails for some bizarre
10424             // reason, we get a predictable result.
10425
10426             info.kp_proc.p_flag = 0;
10427
10428             // Initialize mib, which tells sysctl the info we want, in this case
10429             // we're looking for information about a specific process ID.
10430
10431             mib[0] = CTL_KERN;
10432             mib[1] = KERN_PROC;
10433             mib[2] = KERN_PROC_PID;
10434             mib[3] = getpid();
10435
10436             // Call sysctl.
10437
10438             size = sizeof(info);
10439             if( sysctl(mib, sizeof(mib) / sizeof(*mib), &info, &size, nullptr, 0) != 0 ) {
10440                 Catch::cerr() << "\n** Call to sysctl failed - unable to determine if debugger is
active **\n" << std::endl;
10441                 return false;
10442             }
10443
10444             // We're being debugged if the P_TRACED flag is set.
10445
10446             return ( (info.kp_proc.p_flag & P_TRACED) != 0 );
10447         }
10448     #else
10449         bool isDebuggerActive() {
10450             // We need to find another way to determine this for non-appleclang compilers on macOS
10451             return false;
10452         }
10453     #endif
10454 } // namespace Catch
10455
10456 #elif defined(CATCH_PLATFORM_LINUX)
10457 #include <fstream>
10458 #include <string>
10459
10460 namespace Catch{
10461     // The standard POSIX way of detecting a debugger is to attempt to
10462     // ptrace() the process, but this needs to be done from a child and not
10463     // this process itself to still allow attaching to this process later
10464     // if wanted, so is rather heavy. Under Linux we have the PID of the
10465     // "debugger" (which doesn't need to be gdb, of course, it could also
10466     // be strace, for example) in /proc/$PID/status, so just get it from
10467     // there instead.
10468     bool isDebuggerActive(){
10469         // Libstdc++ has a bug, where std::ifstream sets errno to 0
10470         // This way our users can properly assert over errno values
10471         ErrnoGuard guard;
10472         std::ifstream in("/proc/self/status");
10473         for( std::string line; std::getline(in, line); ) {
10474             static const int PREFIX_LEN = 11;
10475             if( line.compare(0, PREFIX_LEN, "TracerPid:") == 0 ) {
10476                 // We're traced if the PID is not 0 and no other PID starts
10477                 // with 0 digit, so it's enough to check for just a single

```

```

10478             // character.
10479             return line.length() > PREFIX_LEN && line[PREFIX_LEN] != '0';
10480         }
10481     }
10482
10483     return false;
10484 }
10485 // namespace Catch
10486 #elif defined(_MSC_VER)
10487 extern "C" __declspec(dllimport) int __stdcall IsDebuggerPresent();
10488 namespace Catch {
10489     bool isDebuggerActive() {
10490         return IsDebuggerPresent() != 0;
10491     }
10492 }
10493 #elif defined(__MINGW32__)
10494 extern "C" __declspec(dllimport) int __stdcall IsDebuggerPresent();
10495 namespace Catch {
10496     bool isDebuggerActive() {
10497         return IsDebuggerPresent() != 0;
10498     }
10499 }
10500 #else
10501 namespace Catch {
10502     bool isDebuggerActive() { return false; }
10503 }
10504 #endif // Platform
10505 // end catch_debugger.cpp
10506 // start catch_decomposer.cpp
10507
10508 namespace Catch {
10509     ITransientExpression::~ITransientExpression() = default;
10510
10511     void formatReconstructedExpression( std::ostream &os, std::string const& lhs, StringRef op,
10512         std::string const& rhs ) {
10513         if( lhs.size() + rhs.size() < 40 &&
10514             lhs.find('\n') == std::string::npos &&
10515             rhs.find('\n') == std::string::npos )
10516             os << lhs << " " << op << " " << rhs;
10517         else
10518             os << lhs << "\n" << op << "\n" << rhs;
10519     }
10520 }
10521 // end catch_decomposer.cpp
10522 // start catch_enforce.cpp
10523
10524 #include <stdexcept>
10525
10526 namespace Catch {
10527 #if defined(CATCH_CONFIG_DISABLE_EXCEPTIONS) &&
10528     !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS_CUSTOM_HANDLER)
10529     [[noreturn]]
10530     void throw_exception(std::exception const& e) {
10531         Catch::cerr() << "Catch will terminate because it needed to throw an exception.\n"
10532             << "The message was: " << e.what() << '\n';
10533         std::terminate();
10534     }
10535 #endif
10536
10537     [[noreturn]]
10538     void throw_logic_error(std::string const& msg) {
10539         throw_exception(std::logic_error(msg));
10540     }
10541
10542     [[noreturn]]
10543     void throw_domain_error(std::string const& msg) {
10544         throw_exception(std::domain_error(msg));
10545     }
10546
10547     [[noreturn]]
10548     void throw_runtime_error(std::string const& msg) {
10549         throw_exception(std::runtime_error(msg));
10550     }
10551 } // namespace Catch;
10552 // end catch_enforce.cpp
10553 // start catch_enum_values_registry.cpp
10554 // start catch_enum_values_registry.h
10555
10556 #include <vector>
10557 #include <memory>
10558
10559 namespace Catch {
10560     namespace Detail {
10561

```

```

10563         std::unique_ptr<EnumInfo> makeEnumInfo( StringRef enumName, StringRef allValueNames,
10564         std::vector<int> const& values );
10565         class EnumValuesRegistry : public IMutableEnumValuesRegistry {
10566         10567             std::vector<std::unique_ptr<EnumInfo>> m_enumInfos;
10568             EnumInfo const& registerEnum( StringRef enumName, StringRef allEnums, std::vector<int>
10569             const& values) override;
10570         };
10571         std::vector<StringRef> parseEnums( StringRef enums );
10572     } // Detail
10573 } // Catch
10574 // end catch_enum_values_registry.h
10575
10576 #include <map>
10577 #include <cassert>
10578 namespace Catch {
10579     IMutableEnumValuesRegistry::~IMutableEnumValuesRegistry() {}
10580     namespace Detail {
10581         namespace {
10582             // Extracts the actual name part of an enum instance
10583             // In other words, it returns the Blue part of Bikeshed::Colour::Blue
10584             StringRef extractInstanceName(StringRef enumInstance) {
10585                 // Find last occurrence of ":"
10586                 size_t name_start = enumInstance.size();
10587                 while (name_start > 0 && enumInstance[name_start - 1] != ':') {
10588                     --name_start;
10589                 }
10590                 return enumInstance.substr(name_start, enumInstance.size() - name_start);
10591             }
10592         }
10593         std::vector<StringRef> parseEnums( StringRef enums ) {
10594             auto enumValues = splitStringRef( enums, ',' );
10595             std::vector<StringRef> parsed;
10596             parsed.reserve( enumValues.size() );
10597             for( auto const& enumValue : enumValues ) {
10598                 parsed.push_back(trim(extractInstanceName(enumValue)));
10599             }
10600             return parsed;
10601         }
10602         EnumInfo::~EnumInfo() {}
10603         StringRef EnumInfo::lookup( int value ) const {
10604             for( auto const& valueToName : m_values ) {
10605                 if( valueToName.first == value )
10606                     return valueToName.second;
10607             }
10608             return "{** unexpected enum value **}"_sr;
10609         }
10610         std::unique_ptr<EnumInfo> makeEnumInfo( StringRef enumName, StringRef allValueNames,
10611         std::vector<int> const& values ) {
10612             std::unique_ptr<EnumInfo> enumInfo( new EnumInfo );
10613             enumInfo->m_name = enumName;
10614             enumInfo->m_values.reserve( values.size() );
10615             const auto valueNames = Catch::Detail::parseEnums( allValueNames );
10616             assert( valueNames.size() == values.size() );
10617             std::size_t i = 0;
10618             for( auto value : values )
10619                 enumInfo->m_values.emplace_back(value, valueNames[i++]);
10620             return enumInfo;
10621         }
10622         EnumInfo const& EnumValuesRegistry::registerEnum( StringRef enumName, StringRef allValueNames,
10623         std::vector<int> const& values ) {
10624             m_enumInfos.push_back(makeEnumInfo(enumName, allValueNames, values));
10625             return *m_enumInfos.back();
10626         }
10627     } // Detail
10628 } // Catch
10629 // end catch_enum_values_registry.cpp
10630 // start catch_errno_guard.cpp

```

```

10646
10647 #include <cerrno>
10648
10649 namespace Catch {
10650     ErrnoGuard::ErrnoGuard():m_oldErrno(errno){}
10651     ErrnoGuard::~ErrnoGuard() { errno = m_oldErrno; }
10652 }
10653 // end catch_errno_guard.cpp
10654 // start catch_exception_translator_registry.cpp
10655
10656 // start catch_exception_translator_registry.h
10657
10658 #include <vector>
10659 #include <string>
10660 #include <memory>
10661
10662 namespace Catch {
10663
10664     class ExceptionTranslatorRegistry : public IExceptionTranslatorRegistry {
10665     public:
10666         ~ExceptionTranslatorRegistry();
10667         virtual void registerTranslator( const IExceptionTranslator* translator );
10668         std::string translateActiveException() const override;
10669         std::string tryTranslators() const;
10670
10671     private:
10672         std::vector<std::unique_ptr<IExceptionTranslator const> m_translators;
10673     };
10674 }
10675
10676 // end catch_exception_translator_registry.h
10677 #ifdef __OBJC__
10678 #import "Foundation/Foundation.h"
10679 #endif
10680
10681 namespace Catch {
10682
10683     ExceptionTranslatorRegistry::~ExceptionTranslatorRegistry() {
10684     }
10685
10686     void ExceptionTranslatorRegistry::registerTranslator( const IExceptionTranslator* translator ) {
10687         m_translators.push_back( std::unique_ptr<const IExceptionTranslator>( translator ) );
10688     }
10689
10690     #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
10691     std::string ExceptionTranslatorRegistry::translateActiveException() const {
10692         try {
10693             #ifdef __OBJC__
10694                 // In Objective-C try objective-c exceptions first
10695                 @try {
10696                     return tryTranslators();
10697                 }
10698                 @catch (NSEException *exception) {
10699                     return Catch::Detail::stringify( [exception description] );
10700                 }
10701             #else
10702                 // Compiling a mixed mode project with MSVC means that CLR
10703                 // exceptions will be caught in (...) as well. However, these
10704                 // do not fill-in std::current_exception and thus lead to crash
10705                 // when attempting rethrow.
10706                 // /EHa switch also causes structured exceptions to be caught
10707                 // here, but they fill-in current_exception properly, so
10708                 // at worst the output should be a little weird, instead of
10709                 // causing a crash.
10710                 if (std::current_exception() == nullptr) {
10711                     return "Non C++ exception. Possibly a CLR exception.";
10712                 }
10713                 return tryTranslators();
10714             #endif
10715         }
10716         catch( TestFailureException& ) {
10717             std::rethrow_exception(std::current_exception());
10718         }
10719         catch( std::exception& ex ) {
10720             return ex.what();
10721         }
10722         catch( std::string& msg ) {
10723             return msg;
10724         }
10725         catch( const char* msg ) {
10726             return msg;
10727         }
10728         catch(...) {
10729             return "Unknown exception";
10730         }
10731     }
10732

```

```

10733     std::string ExceptionTranslatorRegistry::tryTranslators() const {
10734         if (m_translators.empty()) {
10735             std::rethrow_exception(std::current_exception());
10736         } else {
10737             return m_translators[0]->translate(m_translators.begin() + 1, m_translators.end());
10738         }
10739     }
10740
10741 #else // ^^ Exceptions are enabled // Exceptions are disabled vv
10742     std::string ExceptionTranslatorRegistry::translateActiveException() const {
10743         CATCH_INTERNAL_ERROR("Attempted to translate active exception under
10744         CATCH_CONFIG_DISABLE_EXCEPTIONS!");
10745     }
10746     std::string ExceptionTranslatorRegistry::tryTranslators() const {
10747         CATCH_INTERNAL_ERROR("Attempted to use exception translators under
10748         CATCH_CONFIG_DISABLE_EXCEPTIONS!");
10749     }
10750 #endif
10751 }
10752 // end catch_exception_translator_registry.cpp
10753 // start catch_fatal_condition.cpp
10754
10755 #include <algorithm>
10756
10757 #if !defined( CATCH_CONFIG_WINDOWS_SEH ) && !defined( CATCH_CONFIG_POSIX_SIGNALS )
10758 namespace Catch {
10759
10760     // If neither SEH nor signal handling is required, the handler impls
10761     // do not have to do anything, and can be empty.
10762     void FatalConditionHandler::engage_platform() {}
10763     void FatalConditionHandler::disengage_platform() {}
10764     FatalConditionHandler::FatalConditionHandler() = default;
10765     FatalConditionHandler::~FatalConditionHandler() = default;
10766 } // end namespace Catch
10767
10768 #endif // !CATCH_CONFIG_WINDOWS_SEH && !CATCH_CONFIG_POSIX_SIGNALS
10769
10770 #if defined( CATCH_CONFIG_WINDOWS_SEH ) && defined( CATCH_CONFIG_POSIX_SIGNALS )
10771 #error "Inconsistent configuration: Windows' SEH handling and POSIX signals cannot be enabled at the same time"
10772 #endif // CATCH_CONFIG_WINDOWS_SEH && CATCH_CONFIG_POSIX_SIGNALS
10773
10774 #if defined( CATCH_CONFIG_WINDOWS_SEH ) || defined( CATCH_CONFIG_POSIX_SIGNALS )
10775 namespace {
10776     namespace {
10777         void reportFatal( char const * message ) {
10778             Catch::getCurrentContext().getResultCapture()->handleFatalErrorCondition( message );
10779         }
10780     }
10781     constexpr std::size_t minStackSizeForErrors = 32 * 1024;
10782 } // end unnamed namespace
10783
10784 #endif // CATCH_CONFIG_WINDOWS_SEH || CATCH_CONFIG_POSIX_SIGNALS
10785
10786 #if defined( CATCH_CONFIG_WINDOWS_SEH )
10787 namespace Catch {
10788
10789     struct SignalDefs { DWORD id; const char* name; };
10790
10791     // There is no 1-1 mapping between signals and windows exceptions.
10792     // Windows can easily distinguish between SO and SigSegV,
10793     // but SigInt, SigTerm, etc are handled differently.
10794     static SignalDefs signalDefs[] = {
10795         { static_cast<DWORD>(EXCEPTION_ILLEGAL_INSTRUCTION), "SIGILL - Illegal instruction signal" },
10796         { static_cast<DWORD>(EXCEPTION_STACK_OVERFLOW), "SIGSEGV - Stack overflow" },
10797         { static_cast<DWORD>(EXCEPTION_ACCESS_VIOLATION), "SIGSEGV - Segmentation violation signal" },
10798         { static_cast<DWORD>(EXCEPTION_INT_DIVIDE_BY_ZERO), "Divide by zero error" },
10799     };
10800
10801     static LONG CALLBACK handleVectoredException(PEXCEPTION_POINTERS ExceptionInfo) {
10802         for (auto const& def : signalDefs) {
10803             if (ExceptionInfo->ExceptionCode == def.id) {
10804                 reportFatal(def.name);
10805             }
10806         }
10807         // If its not an exception we care about, pass it along.
10808         // This stops us from eating debugger breaks etc.
10809         return EXCEPTION_CONTINUE_SEARCH;
10810     }
10811
10812     // Since we do not support multiple instantiations, we put these
10813     // into global variables and rely on cleaning them up in outlined

```



```

10821 // constructors/destructors
10822 static PVOID exceptionHandlerHandle = nullptr;
10823
10824 // For MSVC, we reserve part of the stack memory for handling
10825 // memory overflow structured exception.
10826 FatalConditionHandler::FatalConditionHandler() {
10827     ULONG guaranteeSize = static_cast<ULONG>(minStackSizeForErrors);
10828     if (!SetThreadStackGuarantee(&guaranteeSize)) {
10829         // We do not want to fully error out, because needing
10830         // the stack reserve should be rare enough anyway.
10831         Catch::cerr()
10832             << "Failed to reserve piece of stack."
10833             << " Stack overflows will not be reported successfully.";
10834     }
10835 }
10836
10837 // We do not attempt to unset the stack guarantee, because
10838 // Windows does not support lowering the stack size guarantee.
10839 FatalConditionHandler::~FatalConditionHandler() = default;
10840
10841 void FatalConditionHandler::engage_platform() {
10842     // Register as first handler in current chain
10843     exceptionHandlerHandle = AddVectoredExceptionHandler(1, handleVectoredException);
10844     if (!exceptionHandlerHandle) {
10845         CATCH_RUNTIME_ERROR("Could not register vectored exception handler");
10846     }
10847 }
10848
10849 void FatalConditionHandler::disengage_platform() {
10850     if (!RemoveVectoredExceptionHandler(exceptionHandlerHandle)) {
10851         CATCH_RUNTIME_ERROR("Could not unregister vectored exception handler");
10852     }
10853     exceptionHandlerHandle = nullptr;
10854 }
10855
10856 } // end namespace Catch
10857
10858 #endif // CATCH_CONFIG_WINDOWS_SEH
10859
10860 #if defined( CATCH_CONFIG_POSIX_SIGNALS )
10861
10862 #include <signal.h>
10863
10864 namespace Catch {
10865
10866     struct SignalDefs {
10867         int id;
10868         const char* name;
10869     };
10870
10871     static SignalDefs signalDefs[] = {
10872         { SIGINT, "SIGINT - Terminal interrupt signal" },
10873         { SIGILL, "SIGILL - Illegal instruction signal" },
10874         { SIGFPE, "SIGFPE - Floating point error signal" },
10875         { SIGSEGV, "SIGSEGV - Segmentation violation signal" },
10876         { SIGTERM, "SIGTERM - Termination request signal" },
10877         { SIGABRT, "SIGABRT - Abort (abnormal termination) signal" }
10878     };
10879
10880 // Older GCCs trigger -Wmissing-field-initializers for T foo = {}
10881 // which is zero initialization, but not explicit. We want to avoid
10882 // that.
10883 #if defined(__GNUC__)
10884 #    pragma GCC diagnostic push
10885 #    pragma GCC diagnostic ignored "-Wmissing-field-initializers"
10886 #endif
10887
10888     static char* altStackMem = nullptr;
10889     static std::size_t altStackSize = 0;
10890     static stack_t oldSigStack{};
10891     static struct sigaction oldSigActions[sizeof(signalDefs) / sizeof(SignalDefs)]{};
10892
10893     static void restorePreviousSignalHandlers() {
10894         // We set signal handlers back to the previous ones. Hopefully
10895         // nobody overwrote them in the meantime, and doesn't expect
10896         // their signal handlers to live past ours given that they
10897         // installed them after ours..
10898         for (std::size_t i = 0; i < sizeof(signalDefs) / sizeof(SignalDefs); ++i) {
10899             sigaction(signalDefs[i].id, &oldSigActions[i], nullptr);
10900         }
10901         // Return the old stack
10902         sigaltstack(&oldSigStack, nullptr);
10903     }
10904
10905     static void handleSignal( int sig ) {
10906         char const * name = "unknown signal";
10907         for (auto const& def : signalDefs) {

```

```

10908         if (sig == def.id) {
10909             name = def.name;
10910             break;
10911         }
10912     }
10913     // We need to restore previous signal handlers and let them do
10914     // their thing, so that the users can have the debugger break
10915     // when a signal is raised, and so on.
10916     restorePreviousSignalHandlers();
10917     reportFatal( name );
10918     raise( sig );
10919 }
10920
10921 FatalConditionHandler::FatalConditionHandler() {
10922     assert(!altStackMem && "Cannot initialize POSIX signal handler when one already exists");
10923     if (altStackSize == 0) {
10924         altStackSize = std::max(static_cast<size_t>(SIGSTKSZ), minStackSizeForErrors);
10925     }
10926     altStackMem = new char[altStackSize]();
10927 }
10928
10929 FatalConditionHandler::~FatalConditionHandler() {
10930     delete[] altStackMem;
10931     // We signal that another instance can be constructed by zeroing
10932     // out the pointer.
10933     altStackMem = nullptr;
10934 }
10935
10936 void FatalConditionHandler::engage_platform() {
10937     stack_t sigStack;
10938     sigStack.ss_sp = altStackMem;
10939     sigStack.ss_size = altStackSize;
10940     sigStack.ss_flags = 0;
10941     sigaltstack(&sigStack, &oldSigStack);
10942     struct sigaction sa = { };
10943
10944     sa.sa_handler = handleSignal;
10945     sa.sa_flags = SA_ONSTACK;
10946     for (std::size_t i = 0; i < sizeof(signalDefs)/sizeof(SignalDefs); ++i) {
10947         sigaction(signalDefs[i].id, &sa, &oldSigActions[i]);
10948     }
10949 }
10950
10951 #if defined(__GNUC__)
10952 #   pragma GCC diagnostic pop
10953 #endif
10954
10955 void FatalConditionHandler::disengage_platform() {
10956     restorePreviousSignalHandlers();
10957 }
10958
10959 } // end namespace Catch
10960
10961 #endif // CATCH_CONFIG_POSIX_SIGNALS
10962 // end catch_fatal_condition.cpp
10963 // start catch_generators.cpp
10964
10965 #include <limits>
10966 #include <set>
10967
10968 namespace Catch {
10969
10970 IGeneratorTracker::~IGeneratorTracker() {}
10971
10972 const char* GeneratorException::what() const noexcept {
10973     return m_msg;
10974 }
10975
10976 namespace Generators {
10977
10978     GeneratorUntypedBase::~GeneratorUntypedBase() {}
10979
10980     auto acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo ) ->
10981     IGeneratorTracker& {
10982         return getResultCapture().acquireGeneratorTracker( generatorName, lineInfo );
10983     }
10984 } // namespace Generators
10985 } // namespace Catch
10986 // end catch_generators.cpp
10987 // start catch_interfaces_capture.cpp
10988
10989 namespace Catch {
10990     IResultCapture::~IResultCapture() = default;
10991 }
10992 // end catch_interfaces_capture.cpp
10993 // start catch_interfaces_config.cpp

```

```

10994
10995 namespace Catch {
10996     IConfig::~IConfig() = default;
10997 }
10998 // end catch_interfaces_config.cpp
10999 // start catch_interfaces_exception.cpp
11000
11001 namespace Catch {
11002     IExceptionTranslator::~IExceptionTranslator() = default;
11003     IExceptionTranslatorRegistry::~IExceptionTranslatorRegistry() = default;
11004 }
11005 // end catch_interfaces_exception.cpp
11006 // start catch_interfaces_registry_hub.cpp
11007
11008 namespace Catch {
11009     IRegistryHub::~IRegistryHub() = default;
11010     IMutableRegistryHub::~IMutableRegistryHub() = default;
11011 }
11012 // end catch_interfaces_registry_hub.cpp
11013 // start catch_interfaces_reporter.cpp
11014
11015 // start catch_reporter_listening.h
11016
11017 namespace Catch {
11018
11019     class ListeningReporter : public IStreamingReporter {
11020     public:
11021         using Reporters = std::vector<IStreamingReporterPtr>;
11022         Reporters m_listeners;
11023         IStreamingReporterPtr m_reporter = nullptr;
11024         ReporterPreferences m_preferences;
11025
11026         ListeningReporter();
11027
11028         void addListener( IStreamingReporterPtr&& listener );
11029         void addReporter( IStreamingReporterPtr&& reporter );
11030
11031     public: // IStreamingReporter
11032
11033         ReporterPreferences getPreferences() const override;
11034
11035         void noMatchingTestCases( std::string const& spec ) override;
11036
11037         void reportInvalidArguments( std::string const& arg ) override;
11038
11039         static std::set<Verbosity> getSupportedVerbsosities();
11040
11041     #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
11042         void benchmarkPreparing( std::string const& name ) override;
11043         void benchmarkStarting( BenchmarkInfo const& benchmarkInfo ) override;
11044         void benchmarkEnded( BenchmarkStats<> const& benchmarkStats ) override;
11045         void benchmarkFailed( std::string const& ) override;
11046     #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
11047
11048         void testRunStarting( TestRunInfo const& testRunInfo ) override;
11049         void testGroupStarting( GroupInfo const& groupInfo ) override;
11050         void testCaseStarting( TestCaseInfo const& testInfo ) override;
11051         void sectionStarting( SectionInfo const& sectionInfo ) override;
11052         void assertionStarting( AssertionInfo const& assertionInfo ) override;
11053
11054         // The return value indicates if the messages buffer should be cleared:
11055         bool assertionEnded( AssertionStats const& assertionStats ) override;
11056         void sectionEnded( SectionStats const& sectionStats ) override;
11057         void testCaseEnded( TestCaseStats const& testCaseStats ) override;
11058         void testGroupEnded( TestGroupStats const& testGroupStats ) override;
11059         void testRunEnded( TestRunStats const& testRunStats ) override;
11060
11061         void skipTest( TestCaseInfo const& testInfo ) override;
11062         bool isMulti() const override;
11063
11064     };
11065
11066 } // end namespace Catch
11067
11068 // end catch_reporter_listening.h
11069 namespace Catch {
11070
11071     ReporterConfig::ReporterConfig( IConfigPtr const& _fullConfig )
11072     :   m_stream( &_fullConfig->stream() ), m_fullConfig( _fullConfig ) {}
11073
11074     ReporterConfig::ReporterConfig( IConfigPtr const& _fullConfig, std::ostream& _stream )
11075     :   m_stream( _stream ), m_fullConfig( _fullConfig ) {}
11076
11077     std::ostream& ReporterConfig::stream() const { return *m_stream; }
11078     IConfigPtr ReporterConfig::fullConfig() const { return m_fullConfig; }
11079
11080     TestRunInfo::TestRunInfo( std::string const& _name ) : name( _name ) {}

```

```

11081
11082     GroupInfo::GroupInfo( std::string const& _name,
11083                          std::size_t _groupIndex,
11084                          std::size_t _groupsCount )
11085     :   name( _name ),
11086         groupIndex( _groupIndex ),
11087         groupsCounts( _groupsCount )
11088     {}
11089
11090     AssertionStats::AssertionStats( AssertionResult const& _assertionResult,
11091                                     std::vector<MessageInfo> const& _infoMessages,
11092                                     Totals const& _totals )
11093     :   assertionResult( _assertionResult ),
11094         infoMessages( _infoMessages ),
11095         totals( _totals )
11096     {
11097         assertionResult.m_resultData.lazyExpression.m_transientExpression =
11098             _assertionResult.m_resultData.lazyExpression.m_transientExpression;
11099
11100         if( assertionResult.hasMessage() ) {
11101             // Copy message into messages list.
11102             // !TBD This should have been done earlier, somewhere
11103             MessageBuilder builder( assertionResult.getTestMacroName(),
11104                                     assertionResult.getSourceInfo(), assertionResult.getResultType() );
11105             builder « assertionResult.getMessage();
11106             builder.m_info.message = builder.m_stream.str();
11107             infoMessages.push_back( builder.m_info );
11108         }
11109
11110         AssertionStats::~AssertionStats() = default;
11111
11112     SectionStats::SectionStats( SectionInfo const& _sectionInfo,
11113                                Counts const& _assertions,
11114                                double _durationInSeconds,
11115                                bool _missingAssertions )
11116     :   sectionInfo( _sectionInfo ),
11117         assertions( _assertions ),
11118         durationInSeconds( _durationInSeconds ),
11119         missingAssertions( _missingAssertions )
11120     {}
11121
11122     SectionStats::~SectionStats() = default;
11123
11124     TestCaseStats::TestCaseStats( TestCaseInfo const& _testInfo,
11125                                  Totals const& _totals,
11126                                  std::string const& _stdOut,
11127                                  std::string const& _stdErr,
11128                                  bool _aborting )
11129     :   testInfo( _testInfo ),
11130         totals( _totals ),
11131         stdOut( _stdOut ),
11132         stdErr( _stdErr ),
11133         aborting( _aborting )
11134     {}
11135
11136     TestCaseStats::~TestCaseStats() = default;
11137
11138     TestGroupStats::TestGroupStats( GroupInfo const& _groupInfo,
11139                                     Totals const& _totals,
11140                                     bool _aborting )
11141     :   groupInfo( _groupInfo ),
11142         totals( _totals ),
11143         aborting( _aborting )
11144     {}
11145
11146     TestGroupStats::TestGroupStats( GroupInfo const& _groupInfo )
11147     :   groupInfo( _groupInfo ),
11148         aborting( false )
11149     {}
11150
11151     TestGroupStats::~TestGroupStats() = default;
11152
11153     TestRunStats::TestRunStats( TestRunInfo const& _runInfo,
11154                                Totals const& _totals,
11155                                bool _aborting )
11156     :   runInfo( _runInfo ),
11157         totals( _totals ),
11158         aborting( _aborting )
11159     {}
11160
11161     TestRunStats::~TestRunStats() = default;
11162
11163     void IStreamingReporter::fatalErrorEncountered( StringRef ) {}
11164     bool IStreamingReporter::isMulti() const { return false; }
11165

```

```

11166     IReporterFactory::~IReporterFactory() = default;
11167     IReporterRegistry::~IReporterRegistry() = default;
11168
11169 } // end namespace Catch
11170 // end catch_interfaces_reporter.cpp
11171 // start catch_interfaces_runner.cpp
11172
11173 namespace Catch {
11174     IRunner::~IRunner() = default;
11175 }
11176 // end catch_interfaces_runner.cpp
11177 // start catch_interfaces_testcase.cpp
11178
11179 namespace Catch {
11180     ITestInvoker::~ITestInvoker() = default;
11181     ITestCaseRegistry::~ITestCaseRegistry() = default;
11182 }
11183 // end catch_interfaces_testcase.cpp
11184 // start catch_leak_detector.cpp
11185
11186 #ifdef CATCH_CONFIG_WINDOWS_CRTDBG
11187 #include <crtdbg.h>
11188
11189 namespace Catch {
11190
11191     LeakDetector::LeakDetector() {
11192         int flag = _CrtSetDbgFlag(_CRTDBG_REPORT_FLAG);
11193         flag |= _CRTDBG_LEAK_CHECK_DF;
11194         flag |= _CRTDBG_ALLOC_MEM_DF;
11195         _CrtSetDbgFlag(flag);
11196         _CrtSetReportMode(_CRT_WARN, _CRTDBG_MODE_FILE | _CRTDBG_MODE_DEBUG);
11197         _CrtSetReportFile(_CRT_WARN, _CRTDBG_FILE_STDERR);
11198         // Change this to leaking allocation's number to break there
11199         _CrtSetBreakAlloc(-1);
11200     }
11201 }
11202
11203 #else
11204
11205     Catch::LeakDetector::LeakDetector() {}
11206
11207 #endif
11208
11209 Catch::LeakDetector::~LeakDetector() {
11210     Catch::cleanUp();
11211 }
11212 // end catch_leak_detector.cpp
11213 // start catch_list.cpp
11214
11215 // start catch_list.h
11216
11217 #include <set>
11218
11219 namespace Catch {
11220
11221     std::size_t listTests( Config const& config );
11222
11223     std::size_t listTestsNamesOnly( Config const& config );
11224
11225     struct TagInfo {
11226         void add( std::string const& spelling );
11227         std::string all() const;
11228
11229         std::set<std::string> spellings;
11230         std::size_t count = 0;
11231     };
11232
11233     std::size_t listTags( Config const& config );
11234
11235     std::size_t listReporters();
11236
11237     Option<std::size_t> list( std::shared_ptr<Config> const& config );
11238
11239 } // end namespace Catch
11240
11241 // end catch_list.h
11242 // start catch_text.h
11243
11244 namespace Catch {
11245     using namespace clara::TextFlow;
11246 }
11247
11248 // end catch_text.h
11249 #include <limits>
11250 #include <algorithm>
11251 #include <iomanip>
11252

```

```

11253 namespace Catch {
11254
11255     std::size_t listTests( Config const& config ) {
11256         TestSpec const& testSpec = config.testSpec();
11257         if( config.hasTestFilters() )
11258             Catch::cout() << "Matching test cases:\n";
11259         else {
11260             Catch::cout() << "All available test cases:\n";
11261         }
11262
11263         auto matchedTestCases = filterTests( getAllTestCasesSorted( config ), testSpec, config );
11264         for( auto const& testCaseInfo : matchedTestCases ) {
11265             Colour::Code colour = testCaseInfo.isHidden()
11266                 ? Colour::SecondaryText
11267                 : Colour::None;
11268             Colour colourGuard( colour );
11269
11270             Catch::cout() << Column( testCaseInfo.name ).initialIndent( 2 ).indent( 4 ) << "\n";
11271             if( config.verbosity() >= Verbosity::High ) {
11272                 Catch::cout() << Column( Catch::Detail::stringify( testCaseInfo.lineInfo ) ).indent(4)
11273 << std::endl;
11274                 std::string description = testCaseInfo.description;
11275                 if( description.empty() )
11276                     description = "(NO DESCRIPTION)";
11277                 Catch::cout() << Column( description ).indent(4) << std::endl;
11278             }
11279             if( !testCaseInfo.tags.empty() )
11280                 Catch::cout() << Column( testCaseInfo.tagsAsString() ).indent( 6 ) << "\n";
11281         }
11282         if( !config.hasTestFilters() )
11283             Catch::cout() << pluralise( matchedTestCases.size(), "test case" ) << '\n' << std::endl;
11284         else
11285             Catch::cout() << pluralise( matchedTestCases.size(), "matching test case" ) << '\n' <<
11286 std::endl;
11287         return matchedTestCases.size();
11288     }
11289
11290     std::size_t listTestsNamesOnly( Config const& config ) {
11291         TestSpec const& testSpec = config.testSpec();
11292         std::size_t matchedTests = 0;
11293         std::vector<TestCase> matchedTestCases = filterTests( getAllTestCasesSorted( config ),
11294 testSpec, config );
11295         for( auto const& testCaseInfo : matchedTestCases ) {
11296             matchedTests++;
11297             if( startsWith( testCaseInfo.name, '#' ) )
11298                 Catch::cout() << "'" << testCaseInfo.name << "'";
11299             else
11300                 Catch::cout() << testCaseInfo.name;
11301             if ( config.verbosity() >= Verbosity::High )
11302                 Catch::cout() << "\t@" << testCaseInfo.lineInfo;
11303             Catch::cout() << std::endl;
11304         }
11305         return matchedTests;
11306     }
11307
11308     void TagInfo::add( std::string const& spelling ) {
11309         ++count;
11310         spellings.insert( spelling );
11311     }
11312
11313     std::string TagInfo::all() const {
11314         size_t size = 0;
11315         for (auto const& spelling : spellings) {
11316             // Add 2 for the brackets
11317             size += spelling.size() + 2;
11318         }
11319
11320         std::string out; out.reserve(size);
11321         for (auto const& spelling : spellings) {
11322             out += '[';
11323             out += spelling;
11324             out += ']';
11325         }
11326         return out;
11327     }
11328
11329     std::size_t listTags( Config const& config ) {
11330         TestSpec const& testSpec = config.testSpec();
11331         if( config.hasTestFilters() )
11332             Catch::cout() << "Tags for matching test cases:\n";
11333         else {
11334             Catch::cout() << "All available tags:\n";
11335         }
11336
11337         std::map<std::string, TagInfo> tagCounts;
11338     }

```

```

11337         std::vector<TestCase> matchedTestCases = filterTests( getAllTestCasesSorted( config ),
testSpec, config );
11338         for( auto const& testCase : matchedTestCases ) {
11339             for( auto const& tagName : testCase.getTestCaseInfo().tags ) {
11340                 std::string lcaseTagName = toLower( tagName );
11341                 auto countIt = tagCounts.find( lcaseTagName );
11342                 if( countIt == tagCounts.end() )
11343                     countIt = tagCounts.insert( std::make_pair( lcaseTagName, TagInfo() ) ).first;
11344                 countIt->second.add( tagName );
11345             }
11346         }
11347
11348         for( auto const& tagCount : tagCounts ) {
11349             ReusableStringStream rss;
11350             rss << " " << std::setw(2) << tagCount.second.count << " ";
11351             auto str = rss.str();
11352             auto wrapper = Column( tagCount.second.all() )
11353                             .initialIndent( 0 )
11354                             .indent( str.size() )
11355                             .width( CATCH_CONFIG_CONSOLE_WIDTH-10 );
11356             Catch::cout() << str << wrapper << '\n';
11357         }
11358         Catch::cout() << pluralise( tagCounts.size(), "tag" ) << '\n' << std::endl;
11359         return tagCounts.size();
11360     }
11361
11362     std::size_t listReporters() {
11363         Catch::cout() << "Available reporters:\n";
11364         IReporterRegistry::FactoryMap const& factories =
getRegistryHub().getReporterRegistry().getFactories();
11365         std::size_t maxNameLen = 0;
11366         for( auto const& factoryKvp : factories )
11367             maxNameLen = (std::max)( maxNameLen, factoryKvp.first.size() );
11368
11369         for( auto const& factoryKvp : factories ) {
11370             Catch::cout()
11371                 << Column( factoryKvp.first + ":" )
11372                     .indent( 2 )
11373                     .width( 5+maxNameLen )
11374                 + Column( factoryKvp.second->getDescription() )
11375                     .initialIndent( 0 )
11376                     .indent( 2 )
11377                     .width( CATCH_CONFIG_CONSOLE_WIDTH - maxNameLen-8 )
11378                 << "\n";
11379         }
11380         Catch::cout() << std::endl;
11381         return factories.size();
11382     }
11383
11384     Option<std::size_t> list( std::shared_ptr<Config> const& config ) {
11385         Option<std::size_t> listedCount;
11386         getCurrentMutableContext().setConfig( config );
11387         if( config->listTests() )
11388             listedCount = listedCount.valueOr(0) + listTests( *config );
11389         if( config->listTestNamesOnly() )
11390             listedCount = listedCount.valueOr(0) + listTestsNamesOnly( *config );
11391         if( config->listTags() )
11392             listedCount = listedCount.valueOr(0) + listTags( *config );
11393         if( config->listReporters() )
11394             listedCount = listedCount.valueOr(0) + listReporters();
11395         return listedCount;
11396     }
11397
11398 } // end namespace Catch
11399 // end catch_list.cpp
11400 // start catch_matchers.cpp
11401
11402 namespace Catch {
11403     namespace Matchers {
11404         namespace Impl {
11405
11406             std::string MatcherUntypedBase::toString() const {
11407                 if( m_cachedToString.empty() )
11408                     m_cachedToString = describe();
11409                 return m_cachedToString;
11410             }
11411
11412             MatcherUntypedBase::~MatcherUntypedBase() = default;
11413
11414         } // namespace Impl
11415     } // namespace Matchers
11416
11417     using namespace Matchers;
11418     using Matchers::Impl::MatcherBase;
11419
11420 } // namespace Catch
11421 // end catch_matchers.cpp

```

```

11422 // start catch_matchers_exception.cpp
11423
11424 namespace Catch {
11425 namespace Matchers {
11426 namespace Exception {
11427
11428 bool ExceptionMessageMatcher::match(std::exception const& ex) const {
11429     return ex.what() == m_message;
11430 }
11431
11432 std::string ExceptionMessageMatcher::describe() const {
11433     return "exception message matches \"" + m_message + "\"";
11434 }
11435
11436 }
11437
11438 Exception::ExceptionMessageMatcher Message(std::string const& message) {
11439     return Exception::ExceptionMessageMatcher(message);
11440 }
11441 // namespace Exception
11442 } // namespace Matchers
11443 } // namespace Catch
11444 // end catch_matchers_exception.cpp
11445 // start catch_matchers_floating.cpp
11446
11447 // start catch_polyfills.hpp
11448
11449 namespace Catch {
11450     bool isnan(float f);
11451     bool isnan(double d);
11452 }
11453
11454 // end catch_polyfills.hpp
11455 // start catch_to_string.hpp
11456
11457 #include <string>
11458
11459 namespace Catch {
11460     template <typename T>
11461     std::string to_string(T const& t) {
11462 #if defined(CATCH_CONFIG_CPP11_TO_STRING)
11463         return std::to_string(t);
11464 #else
11465         ReusableStringStream rss;
11466         rss << t;
11467         return rss.str();
11468 #endif
11469     }
11470 } // end namespace Catch
11471
11472 // end catch_to_string.hpp
11473 #include <algorithm>
11474 #include <cmath>
11475 #include <cstdlib>
11476 #include <cstdint>
11477 #include <cstring>
11478 #include <sstream>
11479 #include <type_traits>
11480 #include <iomanip>
11481 #include <limits>
11482
11483 namespace Catch {
11484 namespace {
11485
11486     int32_t convert(float f) {
11487         static_assert(sizeof(float) == sizeof(int32_t), "Important ULP matcher assumption violated");
11488         int32_t i;
11489         std::memcpy(&i, &f, sizeof(f));
11490         return i;
11491     }
11492
11493     int64_t convert(double d) {
11494         static_assert(sizeof(double) == sizeof(int64_t), "Important ULP matcher assumption violated");
11495         int64_t i;
11496         std::memcpy(&i, &d, sizeof(d));
11497         return i;
11498     }
11499
11500     template <typename FP>
11501     bool almostEqualUlp(FP lhs, FP rhs, uint64_t maxUlpDiff) {
11502         // Comparison with NaN should always be false.
11503         // This way we can rule it out before getting into the ugly details
11504         if (Catch::isnan(lhs) || Catch::isnan(rhs)) {
11505             return false;
11506         }
11507
11508         auto lc = convert(lhs);

```



```

11509         auto rc = convert(rhs);
11510
11511         if ((lc < 0) != (rc < 0)) {
11512             // Potentially we can have +0 and -0
11513             return lhs == rhs;
11514         }
11515
11516         // static cast as a workaround for IBM XLC
11517         auto ulpDiff = std::abs(static_cast<FP>(lc - rc));
11518         return static_cast<uint64_t>(ulpDiff) <= maxUlpDiff;
11519     }
11520
11521 #if defined(CATCH_CONFIG_GLOBAL_NEXTAFTER)
11522
11523     float nextafter(float x, float y) {
11524         return ::nextafterf(x, y);
11525     }
11526
11527     double nextafter(double x, double y) {
11528         return ::nextafter(x, y);
11529     }
11530
11531 #endif // ^^^ CATCH_CONFIG_GLOBAL_NEXTAFTER ^^^
11532
11533 template <typename FP>
11534 FP step(FP start, FP direction, uint64_t steps) {
11535     for (uint64_t i = 0; i < steps; ++i) {
11536         #if defined(CATCH_CONFIG_GLOBAL_NEXTAFTER)
11537             start = Catch::nextafter(start, direction);
11538         #else
11539             start = std::nextafter(start, direction);
11540         #endif
11541     }
11542     return start;
11543 }
11544
11545 // Performs equivalent check of std::fabs(lhs - rhs) <= margin
11546 // But without the subtraction to allow for INFINITY in comparison
11547 bool marginComparison(double lhs, double rhs, double margin) {
11548     return (lhs + margin >= rhs) && (rhs + margin >= lhs);
11549 }
11550
11551 template <typename FloatingPoint>
11552 void write(std::ostream& out, FloatingPoint num) {
11553     out << std::scientific
11554         << std::setprecision(std::numeric_limits<FloatingPoint>::max_digits10 - 1)
11555         << num;
11556 }
11557
11558 } // end anonymous namespace
11559
11560 namespace Matchers {
11561 namespace Floating {
11562
11563     enum class FloatingPointKind : uint8_t {
11564         Float,
11565         Double
11566     };
11567
11568     WithinAbsMatcher::WithinAbsMatcher(double target, double margin)
11569         : m_target{ target }, m_margin{ margin } {
11570         CATCH_ENFORCE(margin >= 0, "Invalid margin: " << margin << "'.'
11571             << " Margin has to be non-negative.");
11572     }
11573
11574     // Performs equivalent check of std::fabs(lhs - rhs) <= margin
11575     // But without the subtraction to allow for INFINITY in comparison
11576     bool WithinAbsMatcher::match(double const& matchee) const {
11577         return (matchee + m_margin >= m_target) && (m_target + m_margin >= matchee);
11578     }
11579
11580     std::string WithinAbsMatcher::describe() const {
11581         return "is within " + ::Catch::Detail::stringify(m_margin) + " of " +
11582             ::Catch::Detail::stringify(m_target);
11583     }
11584
11585     WithinUlpMatcher::WithinUlpMatcher(double target, uint64_t ulps, FloatingPointKind baseType)
11586         : m_target{ target }, m_ulps{ ulps }, m_type{ baseType } {
11587         CATCH_ENFORCE(m_type == FloatingPointKind::Double
11588             || m_ulps < (std::numeric_limits<uint32_t>::max)(),
11589             "Provided ULP is impossibly large for a float comparison.");
11589     }
11590
11591 #if defined(__clang__)
11592 #pragma clang diagnostic push
11593 // Clang <3.5 reports on the default branch in the switch below
11594 #pragma clang diagnostic ignored "-Wunreachable-code"

```

```

11595 #endif
11596
11597     bool WithinUlpMatcher::match(double const& matchee) const {
11598         switch (m_type) {
11599             case FloatingPointKind::Float:
11600                 return almostEqualUlp<float>(static_cast<float>(matchee), static_cast<float>(m_target),
m_ulps);
11601             case FloatingPointKind::Double:
11602                 return almostEqualUlp<double>(matchee, m_target, m_ulps);
11603             default:
11604                 CATCH_INTERNAL_ERROR( "Unknown FloatingPointKind value" );
11605         }
11606     }
11607
11608 #if defined(__clang__)
11609 #pragma clang diagnostic pop
11610 #endif
11611
11612     std::string WithinUlpMatcher::describe() const {
11613         std::stringstream ret;
11614
11615         ret << "is within " << m_ulps << " ULPs of ";
11616
11617         if (m_type == FloatingPointKind::Float) {
11618             write(ret, static_cast<float>(m_target));
11619             ret << 'f';
11620         } else {
11621             write(ret, m_target);
11622         }
11623
11624         ret << " ([";
11625         if (m_type == FloatingPointKind::Double) {
11626             write(ret, step(m_target, static_cast<double>(-INFINITY), m_ulps));
11627             ret << ", ";
11628             write(ret, step(m_target, static_cast<double>( INFINITY), m_ulps));
11629         } else {
11630             // We have to cast INFINITY to float because of MinGW, see #1782
11631             write(ret, step(static_cast<float>(m_target), static_cast<float>(-INFINITY), m_ulps));
11632             ret << ", ";
11633             write(ret, step(static_cast<float>(m_target), static_cast<float>( INFINITY), m_ulps));
11634         }
11635         ret << "])";
11636
11637         return ret.str();
11638     }
11639
11640     WithinRelMatcher::WithinRelMatcher(double target, double epsilon):
11641         m_target(target),
11642         m_epsilon(epsilon){
11643         CATCH_ENFORCE(m_epsilon >= 0., "Relative comparison with epsilon < 0 does not make sense.");
11644         CATCH_ENFORCE(m_epsilon < 1., "Relative comparison with epsilon >= 1 does not make sense.");
11645     }
11646
11647     bool WithinRelMatcher::match(double const& matchee) const {
11648         const auto relMargin = m_epsilon * (std::max)(std::fabs(matchee), std::fabs(m_target));
11649         return marginComparison(matchee, m_target,
11650                                std::isinf(relMargin)? 0 : relMargin);
11651     }
11652
11653     std::string WithinRelMatcher::describe() const {
11654         Catch::ReusableStringStream sstr;
11655         sstr << "and " << m_target << " are within " << m_epsilon * 100. << "% of each other";
11656         return sstr.str();
11657     }
11658
11659 } // namespace Floating
11660
11661 Floating::WithinUlpMatcher WithinULP(double target, uint64_t maxUlpDiff) {
11662     return Floating::WithinUlpMatcher(target, maxUlpDiff, Floating::FloatingPointKind::Double);
11663 }
11664
11665 Floating::WithinUlpMatcher WithinULP(float target, uint64_t maxUlpDiff) {
11666     return Floating::WithinUlpMatcher(target, maxUlpDiff, Floating::FloatingPointKind::Float);
11667 }
11668
11669 Floating::WithinAbsMatcher WithinAbs(double target, double margin) {
11670     return Floating::WithinAbsMatcher(target, margin);
11671 }
11672
11673 Floating::WithinRelMatcher WithinRel(double target, double eps) {
11674     return Floating::WithinRelMatcher(target, eps);
11675 }
11676
11677 Floating::WithinRelMatcher WithinRel(double target) {
11678     return Floating::WithinRelMatcher(target, std::numeric_limits<double>::epsilon() * 100);
11679 }
11680

```

```

11681 Floating::WithinRelMatcher WithinRel(float target, float eps) {
11682     return Floating::WithinRelMatcher(target, eps);
11683 }
11684
11685 Floating::WithinRelMatcher WithinRel(float target) {
11686     return Floating::WithinRelMatcher(target, std::numeric_limits<float>::epsilon() * 100);
11687 }
11688
11689 } // namespace Matchers
11690 } // namespace Catch
11691 // end catch_matchers_floating.cpp
11692 // start catch_matchers_generic.cpp
11693
11694 std::string Catch::Matchers::Generic::Detail::finalizeDescription(const std::string& desc) {
11695     if (desc.empty()) {
11696         return "matches undescribed predicate";
11697     } else {
11698         return "matches predicate: \"" + desc + '"';
11699     }
11700 }
11701 // end catch_matchers_generic.cpp
11702 // start catch_matchers_string.cpp
11703
11704 #include <regex>
11705
11706 namespace Catch {
11707 namespace Matchers {
11708     namespace StdString {
11709         CasedString::CasedString( std::string const& str, CaseSensitive::Choice caseSensitivity )
11710         :   m_caseSensitivity( caseSensitivity ),
11711             m_str( adjustString( str ) )
11712         {}
11713         std::string CasedString::adjustString( std::string const& str ) const {
11714             return m_caseSensitivity == CaseSensitive::No
11715                 ? toLower( str )
11716                 : str;
11717         }
11718         std::string CasedString::caseSensitivitySuffix() const {
11719             return m_caseSensitivity == CaseSensitive::No
11720                 ? " (case insensitive)"
11721                 : std::string();
11722         }
11723     }
11724
11725     StringMatcherBase::StringMatcherBase( std::string const& operation, CasedString const&
11726 comparator )
11727 : m_comparator( comparator ),
11728   m_operation( operation ) {}
11729
11730     std::string StringMatcherBase::describe() const {
11731         std::string description;
11732         description.reserve(5 + m_operation.size() + m_comparator.m_str.size() +
11733                             m_comparator.caseSensitivitySuffix().size());
11734         description += m_operation;
11735         description += ": ";
11736         description += m_comparator.m_str;
11737         description += "\n";
11738         description += m_comparator.caseSensitivitySuffix();
11739         return description;
11740     }
11741
11742     EqualsMatcher::EqualsMatcher( CasedString const& comparator ) : StringMatcherBase( "equals",
11743 comparator ) {}
11744
11745     bool EqualsMatcher::match( std::string const& source ) const {
11746         return m_comparator.adjustString( source ) == m_comparator.m_str;
11747     }
11748
11749     ContainsMatcher::ContainsMatcher( CasedString const& comparator ) : StringMatcherBase(
11750 "contains", comparator ) {}
11751
11752     bool ContainsMatcher::match( std::string const& source ) const {
11753         return contains( m_comparator.adjustString( source ), m_comparator.m_str );
11754     }
11755
11756     StartsWithMatcher::StartsWithMatcher( CasedString const& comparator ) : StringMatcherBase(
11757 "starts with", comparator ) {}
11758
11759     bool StartsWithMatcher::match( std::string const& source ) const {
11760         return startsWith( m_comparator.adjustString( source ), m_comparator.m_str );
11761     }
11762
11763     EndsWithMatcher::EndsWithMatcher( CasedString const& comparator ) : StringMatcherBase( "ends
11764 with", comparator ) {}

```

```

11763         bool EndsWithMatcher::match( std::string const& source ) const {
11764             return endsWith( m_comparator.adjustString( source ), m_comparator.m_str );
11765         }
11766
11767         RegexMatcher::RegexMatcher(std::string regex, CaseSensitive::Choice caseSensitivity):
11768             m_regex(std::move(regex)), m_caseSensitivity(caseSensitivity) {}
11769
11770         bool RegexMatcher::match(std::string const& matchee) const {
11771             auto flags = std::regex::ECMAScript; // ECMAScript is the default syntax option anyway
11772             if (m_caseSensitivity == CaseSensitive::Choice::No) {
11773                 flags |= std::regex::icase;
11774             }
11775             auto reg = std::regex(m_regex, flags);
11776             return std::regex_match(matchee, reg);
11777         }
11778
11779         std::string RegexMatcher::describe() const {
11780             return "matches " + ::Catch::Detail::stringify(m_regex) + ((m_caseSensitivity ==
11781             CaseSensitive::Choice::Yes)? " case sensitively" : " case insensitively");
11782         }
11783     } // namespace StdString
11784
11785     StdString::EqualsMatcher Equals( std::string const& str, CaseSensitive::Choice caseSensitivity ) {
11786         return StdString::EqualsMatcher( StdString::CasedString( str, caseSensitivity) );
11787     }
11788     StdString::ContainsMatcher Contains( std::string const& str, CaseSensitive::Choice caseSensitivity
11789 ) {
11790         return StdString::ContainsMatcher( StdString::CasedString( str, caseSensitivity) );
11791     }
11792     StdString::EndsWithMatcher EndsWith( std::string const& str, CaseSensitive::Choice caseSensitivity
11793 ) {
11794         return StdString::EndsWithMatcher( StdString::CasedString( str, caseSensitivity) );
11795     }
11796     StdString::StartsWithMatcher StartsWith( std::string const& str, CaseSensitive::Choice
11797 caseSensitivity ) {
11798         return StdString::StartsWithMatcher( StdString::CasedString( str, caseSensitivity) );
11799     }
11800
11801     StdString::RegexMatcher Matches(std::string const& regex, CaseSensitive::Choice caseSensitivity) {
11802         return StdString::RegexMatcher(regex, caseSensitivity);
11803     }
11804 } // namespace Matchers
11805 } // namespace Catch
11806 // end catch_matchers_string.cpp
11807 // start catch_message.cpp
11808
11809 namespace Catch {
11810     bool uncaught_exceptions();
11811 } // end namespace Catch
11812
11813 // end catch_uncaught_exceptions.h
11814 #include <cassert>
11815 #include <stack>
11816
11817 namespace Catch {
11818     MessageInfo::MessageInfo( StringRef const& _macroName,
11819                               SourceLineInfo const& _lineInfo,
11820                               ResultWas::OfType _type )
11821     :   macroName( _macroName ),
11822         lineInfo( _lineInfo ),
11823         type( _type ),
11824         sequence( ++globalCount )
11825     {}
11826
11827     bool MessageInfo::operator==( MessageInfo const& other ) const {
11828         return sequence == other.sequence;
11829     }
11830
11831     bool MessageInfo::operator<( MessageInfo const& other ) const {
11832         return sequence < other.sequence;
11833     }
11834
11835     // This may need protecting if threading support is added
11836     unsigned int MessageInfo::globalCount = 0;
11837
11838     Catch::MessageBuilder::MessageBuilder( StringRef const& macroName,
11839                                             SourceLineInfo const& lineInfo,
11840                                             ResultWas::OfType type )
11841     :   m_info(macroName, lineInfo, type) {}
11842
11843     Catch::MessageBuilder::operator MessageInfo() const {
11844         return m_info;
11845     }
11846

```

```

11847     ScopedMessage::ScopedMessage( MessageBuilder const& builder )
11848     : m_info( builder.m_info ), m_moved()
11849     {
11850         m_info.message = builder.m_stream.str();
11851         getResultCapture().pushScopedMessage( m_info );
11852     }
11853
11854     ScopedMessage::ScopedMessage( ScopedMessage&& old )
11855     : m_info( old.m_info ), m_moved()
11856     {
11857         old.m_moved = true;
11858     }
11859
11860     ScopedMessage::~ScopedMessage() {
11861         if ( !uncaught_exceptions() && !m_moved ){
11862             getResultCapture().popScopedMessage(m_info);
11863         }
11864     }
11865
11866     Capturer::Capturer( StringRef macroName, SourceLineInfo const& lineInfo, ResultWas::OfType
resultType, StringRef names ) {
11867         auto trimmed = [&] (size_t start, size_t end) {
11868             while (names[start] == ',' || isspace(static_cast<unsigned char>(names[start]))) {
11869                 ++start;
11870             }
11871             while (names[end] == ',' || isspace(static_cast<unsigned char>(names[end]))) {
11872                 --end;
11873             }
11874             return names.substr(start, end - start + 1);
11875         };
11876         auto skipq = [&] (size_t start, char quote) {
11877             for (auto i = start + 1; i < names.size(); ++i) {
11878                 if (names[i] == quote)
11879                     return i;
11880                 if (names[i] == '\\')
11881                     ++i;
11882             }
11883             CATCH_INTERNAL_ERROR("CAPTURE parsing encountered unmatched quote");
11884         };
11885
11886         size_t start = 0;
11887         std::stack<char> openings;
11888         for (size_t pos = 0; pos < names.size(); ++pos) {
11889             char c = names[pos];
11890             switch (c) {
11891                 case '[':
11892                 case '{':
11893                 case '(':
11894                     // It is basically impossible to disambiguate between
11895                     // comparison and start of template args in this context
11896                     // case '<':
11897                     openings.push(c);
11898                     break;
11899                 case ']':
11900                 case '}':
11901                 case ')':
11902                     // case '>':
11903                     openings.pop();
11904                     break;
11905                 case '"':
11906                 case '\'':
11907                     pos = skipq(pos, c);
11908                     break;
11909                 case ',':
11910                     if (start != pos && openings.empty()) {
11911                         m_messages.emplace_back(macroName, lineInfo, resultType);
11912                         m_messages.back().message = static_cast<std::string>(trimmed(start, pos));
11913                         m_messages.back().message += " := ";
11914                         start = pos;
11915                     }
11916             }
11917         }
11918         assert(openings.empty() && "Mismatched openings");
11919         m_messages.emplace_back(macroName, lineInfo, resultType);
11920         m_messages.back().message = static_cast<std::string>(trimmed(start, names.size() - 1));
11921         m_messages.back().message += " := ";
11922     }
11923     Capturer::~Capturer() {
11924         if ( !uncaught_exceptions() ){
11925             assert( m_captured == m_messages.size() );
11926             for( size_t i = 0; i < m_captured; ++i )
11927                 m_resultCapture.popScopedMessage( m_messages[i] );
11928         }
11929     }
11930
11931     void Capturer::captureValue( size_t index, std::string const& value ) {
11932         assert( index < m_messages.size() );

```

```

11933         m_messages[index].message += value;
11934         m_resultCapture.pushScopedMessage( m_messages[index] );
11935         m_captured++;
11936     }
11937
11938 } // end namespace Catch
11939 // end catch_message.cpp
11940 // start catch_output_redirect.cpp
11941
11942 // start catch_output_redirect.h
11943 #ifndef TWOBLUECUBES_CATCH_OUTPUT_REDIRECT_H
11944 #define TWOBLUECUBES_CATCH_OUTPUT_REDIRECT_H
11945
11946 #include <cstdio>
11947 #include <iosfwd>
11948 #include <string>
11949
11950 namespace Catch {
11951
11952     class RedirectedStream {
11953     public:
11954         RedirectedStream( std::ostream& originalStream, std::ostream& redirectionStream );
11955         ~RedirectedStream();
11956     };
11957
11958     class RedirectedStdOut {
11959     public:
11960         RedirectedStdOut();
11961         auto str() const -> std::string;
11962     };
11963
11964     // StdErr has two constituent streams in C++, std::cerr and std::clog
11965     // This means that we need to redirect 2 streams into 1 to keep proper
11966     // order of writes
11967     class RedirectedStdErr {
11968     public:
11969         RedirectedStdErr();
11970         auto str() const -> std::string;
11971     };
11972
11973     class RedirectedStreams {
11974     public:
11975         RedirectedStreams(RedirectedStreams const&) = delete;
11976         RedirectedStreams& operator=(RedirectedStreams const&) = delete;
11977         RedirectedStreams(RedirectedStreams&&) = delete;
11978         RedirectedStreams& operator=(RedirectedStreams&&) = delete;
11979
11980         RedirectedStreams(std::string& redirectedCout, std::string& redirectedCerr);
11981         ~RedirectedStreams();
11982     private:
11983         std::string& m_redirectedCout;
11984         std::string& m_redirectedCerr;
11985         RedirectedStdOut m_redirectedStdOut;
11986         RedirectedStdErr m_redirectedStdErr;
11987     };
11988
11989 #if defined(CATCH_CONFIG_NEW_CAPTURE)
11990
11991     // Windows's implementation of std::tmpfile is terrible (it tries
11992     // to create a file inside system folder, thus requiring elevated
11993     // privileges for the binary), so we have to use tmpnam(_s) and
11994     // create the file ourselves there.
11995     class TempFile {
11996     public:
11997         TempFile(TempFile const&) = delete;
11998         TempFile& operator=(TempFile const&) = delete;
11999         TempFile(TempFile&&) = delete;
12000         TempFile& operator=(TempFile&&) = delete;
12001
12002         TempFile();
12003         ~TempFile();
12004
12005         std::FILE* getFile();
12006         std::string getContents();
12007     private:
12008         std::FILE* m_file = nullptr;
12009     };
12010 #endif

```

```

12020     char m_buffer[L_tmpnam] = { 0 };
12021 #endif
12022 };
12023
12024 class OutputRedirect {
12025 public:
12026     OutputRedirect(OutputRedirect const&) = delete;
12027     OutputRedirect& operator=(OutputRedirect const&) = delete;
12028     OutputRedirect(OutputRedirect&&) = delete;
12029     OutputRedirect& operator=(OutputRedirect&&) = delete;
12030
12031     OutputRedirect(std::string& stdout_dest, std::string& stderr_dest);
12032     ~OutputRedirect();
12033
12034 private:
12035     int m_originalStdout = -1;
12036     int m_originalStderr = -1;
12037     TempFile m_stdoutFile;
12038     TempFile m_stderrFile;
12039     std::string& m_stdoutDest;
12040     std::string& m_stderrDest;
12041 };
12042 #endif
12043 // end namespace Catch
12044
12045 #endif // TWOBLUECUBES_CATCH_OUTPUT_REDIRECT_H
12046 // end catch_output_redirect.h
12047 #include <cstdio>
12048 #include <cstring>
12049 #include <fstream>
12050 #include <sstream>
12051 #include <stdexcept>
12052
12053 #if defined(CATCH_CONFIG_NEW_CAPTURE)
12054 #if defined(_MSC_VER)
12055 #include <io.h> // _dup and _dup2
12056 #define dup _dup
12057 #define dup2 _dup2
12058 #define fileno _fileno
12059 #else
12060 #include <unistd.h> // dup and dup2
12061 #endif
12062 #endif
12063 #endif
12064 namespace Catch {
12065
12066     RedirectedStream::RedirectedStream( std::ostream& originalStream, std::ostream& redirectionStream
12067 )
12068 :   m_originalStream( originalStream ),
12069     m_redirectionStream( redirectionStream ),
12070     m_prevBuf( m_originalStream.rdbuf() )
12071 {
12072     m_originalStream.rdbuf( m_redirectionStream.rdbuf() );
12073 }
12074
12075 RedirectedStream::~RedirectedStream() {
12076     m_originalStream.rdbuf( m_prevBuf );
12077 }
12078
12079 RedirectedStdOut::RedirectedStdOut() : m_cout( Catch::cout(), m_rss.get() ) {}
12080 auto RedirectedStdOut::str() const -> std::string { return m_rss.str(); }
12081
12082 RedirectedStdErr::RedirectedStdErr()
12083 :   m_cerr( Catch::cerr(), m_rss.get() ),
12084     m_clog( Catch::clog(), m_rss.get() )
12085 {}
12086 auto RedirectedStdErr::str() const -> std::string { return m_rss.str(); }
12087
12088 RedirectedStreams::RedirectedStreams(std::string& redirectedCout, std::string& redirectedCerr)
12089 :   m_redirectedCout( redirectedCout ),
12090     m_redirectedCerr( redirectedCerr )
12091 {}
12092
12093 RedirectedStreams::~RedirectedStreams() {
12094     m_redirectedCout += m_redirectedStdOut.str();
12095     m_redirectedCerr += m_redirectedStdErr.str();
12096 }
12097
12098 #if defined(CATCH_CONFIG_NEW_CAPTURE)
12099 #if defined(_MSC_VER)
12100 TempFile::TempFile() {
12101     if (tmpnam_s(m_buffer)) {
12102         CATCH_RUNTIME_ERROR("Could not get a temp filename");
12103     }
12104 }

```

```

12106         if (fopen_s(&m_file, m_buffer, "w+")) {
12107             char buffer[100];
12108             if (strerror_s(buffer, errno)) {
12109                 CATCH_RUNTIME_ERROR("Could not translate errno to a string");
12110             }
12111             CATCH_RUNTIME_ERROR("Could not open the temp file: '" « m_buffer « "' because: " «
buffer);
12112         }
12113     }
12114 #else
12115     TempFile::TempFile() {
12116         m_file = std::tmpfile();
12117         if (!m_file) {
12118             CATCH_RUNTIME_ERROR("Could not create a temp file.");
12119         }
12120     }
12121
12122 #endif
12123
12124     TempFile::~TempFile() {
12125         // TBD: What to do about errors here?
12126         std::fclose(m_file);
12127         // We manually create the file on Windows only, on Linux
12128         // it will be autodeleted
12129 #if defined(_MSC_VER)
12130         std::remove(m_buffer);
12131 #endif
12132     }
12133
12134     FILE* TempFile::getFile() {
12135         return m_file;
12136     }
12137
12138     std::string TempFile::getContents() {
12139         std::stringstream sstr;
12140         char buffer[100] = {};
12141         std::rewind(m_file);
12142         while (std::fgets(buffer, sizeof(buffer), m_file)) {
12143             sstr « buffer;
12144         }
12145         return sstr.str();
12146     }
12147
12148     OutputRedirect::OutputRedirect(std::string& stdout_dest, std::string& stderr_dest) :
12149         m_originalStdout(dup(1)),
12150         m_originalStderr(dup(2)),
12151         m_stdoutDest(stdout_dest),
12152         m_stderrDest(stderr_dest) {
12153         dup2(fileno(m_stdoutFile.getFile()), 1);
12154         dup2(fileno(m_stderrFile.getFile()), 2);
12155     }
12156
12157     OutputRedirect::~OutputRedirect() {
12158         Catch::cout() « std::flush;
12159         fflush(stdout);
12160         // Since we support overriding these streams, we flush cerr
12161         // even though std::cerr is unbuffered
12162         Catch::cerr() « std::flush;
12163         Catch::clog() « std::flush;
12164         fflush(stderr);
12165
12166         dup2(m_originalStdout, 1);
12167         dup2(m_originalStderr, 2);
12168
12169         m_stdoutDest += m_stdoutFile.getContents();
12170         m_stderrDest += m_stderrFile.getContents();
12171     }
12172
12173 #endif // CATCH_CONFIG_NEW_CAPTURE
12174
12175 } // namespace Catch
12176
12177 #if defined(CATCH_CONFIG_NEW_CAPTURE)
12178 #if defined(_MSC_VER)
12179 #undef dup
12180 #undef dup2
12181 #undef fileno
12182 #endif
12183 #endif
12184 // end catch_output_redirect.cpp
12185 // start catch_polyfills.cpp
12186
12187 #include <cmath>
12188
12189 namespace Catch {
12190
12191 #if !defined(CATCH_CONFIG_POLYFILL_ISNAN)

```



```

12192     bool isnan(float f) {
12193         return std::isnan(f);
12194     }
12195     bool isnan(double d) {
12196         return std::isnan(d);
12197     }
12198 #else
12199     // For now we only use this for embarcadero
12200     bool isnan(float f) {
12201         return std::_isnan(f);
12202     }
12203     bool isnan(double d) {
12204         return std::_isnan(d);
12205     }
12206 #endif
12207
12208 } // end namespace Catch
12209 // end catch_polyfills.cpp
12210 // start catch_random_number_generator.cpp
12211
12212 namespace Catch {
12213     namespace {
12214         #if defined(_MSC_VER)
12215         #pragma warning(push)
12216         #pragma warning(disable:4146) // we negate uint32 during the rotate
12217         #endif
12218         // Safe rotr implementation thanks to John Regehr
12219         uint32_t rotate_right(uint32_t val, uint32_t count) {
12220             const uint32_t mask = 31;
12221             count &= mask;
12222             return (val >> count) | (val << (-count & mask));
12223         }
12224         #if defined(_MSC_VER)
12225         #pragma warning(pop)
12226         #endif
12227     }
12228
12229     SimplePcg32::SimplePcg32(result_type seed_) {
12230         seed(seed_);
12231     }
12232
12233     void SimplePcg32::seed(result_type seed_) {
12234         m_state = 0;
12235         (*this)();
12236         m_state += seed_;
12237         (*this)();
12238     }
12239
12240     void SimplePcg32::discard(uint64_t skip) {
12241         // We could implement this to run in O(log n) steps, but this
12242         // should suffice for our use case.
12243         for (uint64_t s = 0; s < skip; ++s) {
12244             static_cast<void>((*this)());
12245         }
12246     }
12247
12248     SimplePcg32::result_type SimplePcg32::operator()() {
12249         // prepare the output value
12250         const uint32_t xorshifted = static_cast<uint32_t>((m_state >> 18u) ^ m_state) >> 27u;
12251         const auto output = rotate_right(xorshifted, m_state >> 59u);
12252
12253         // advance state
12254         m_state = m_state * 6364136223846793005ULL + s_inc;
12255
12256         return output;
12257     }
12258
12259     bool operator==(SimplePcg32 const& lhs, SimplePcg32 const& rhs) {
12260         return lhs.m_state == rhs.m_state;
12261     }
12262
12263     bool operator!=(SimplePcg32 const& lhs, SimplePcg32 const& rhs) {
12264         return lhs.m_state != rhs.m_state;
12265     }
12266
12267 // end catch_random_number_generator.cpp
12268 // start catch_registry_hub.cpp
12269 // start catch_test_case_registry_impl.h
12270
12271 #include <vector>
12272 #include <set>
12273 #include <algorithm>

```

```

12279 #include <ios>
12280
12281 namespace Catch {
12282     class TestCase;
12283     struct IConfig;
12284
12285     std::vector<TestCase> sortTests( IConfig const& config, std::vector<TestCase> const&
12286         unsortedTestCases );
12287
12288     bool isThrowSafe( TestCase const& testCase, IConfig const& config );
12289     bool matchTest( TestCase const& testCase, TestSpec const& testSpec, IConfig const& config );
12290
12291     void enforceNoDuplicateTestCases( std::vector<TestCase> const& functions );
12292
12293     std::vector<TestCase> filterTests( std::vector<TestCase> const& testCases, TestSpec const&
12294         testSpec, IConfig const& config );
12295     std::vector<TestCase> const& getAllTestCasesSorted( IConfig const& config );
12296
12297     class TestRegistry : public ITestRegistry {
12298     public:
12299         virtual ~TestRegistry() = default;
12300
12301         virtual void registerTest( TestCase const& testCase );
12302
12303         std::vector<TestCase> const& getAllTests() const override;
12304         std::vector<TestCase> const& getAllTestsSorted( IConfig const& config ) const override;
12305
12306     private:
12307         std::vector<TestCase> m_functions;
12308         mutable RunTests::InWhatOrder m_currentSortOrder = RunTests::InDeclarationOrder;
12309         mutable std::vector<TestCase> m_sortedFunctions;
12310         std::size_t m_unnamedCount = 0;
12311         std::ios_base::Init m_ostreamInit; // Forces cout/ cerr to be initialised
12312     };
12313
12314     class TestInvokerAsFunction : public ITestInvoker {
12315     public:
12316         void(*m_testAsFunction)();
12317
12318         TestInvokerAsFunction( void(*testAsFunction)() ) noexcept;
12319
12320         void invoke() const override;
12321     };
12322
12323     std::string extractClassName( StringRef const& classOrQualifiedMethodName );
12324
12325 } // end namespace Catch
12326
12327 // end catch_test_case_registry_impl.h
12328 // start catch_reporter_registry.h
12329 #include <map>
12330
12331 namespace Catch {
12332     class ReporterRegistry : public IReporterRegistry {
12333     public:
12334         ~ReporterRegistry() override;
12335
12336         IStreamingReporterPtr create( std::string const& name, IConfigPtr const& config ) const
12337             override;
12338
12339         void registerReporter( std::string const& name, IReporterFactoryPtr const& factory );
12340         void registerListener( IReporterFactoryPtr const& factory );
12341
12342         FactoryMap const& getFactories() const override;
12343         Listeners const& getListeners() const override;
12344
12345     private:
12346         FactoryMap m_factories;
12347         Listeners m_listeners;
12348     };
12349
12350 // end catch_reporter_registry.h
12351 // start catch_tag_alias_registry.h
12352 // start catch_tag_alias.h
12353 #include <string>
12354
12355 namespace Catch {

```

```

12365     struct TagAlias {
12366         TagAlias(std::string const& _tag, SourceLineInfo _lineInfo);
12367
12368         std::string tag;
12369         SourceLineInfo lineInfo;
12370     };
12371
12372 } // end namespace Catch
12373
12374 // end catch_tag_alias.h
12375 #include <map>
12376
12377 namespace Catch {
12378
12379     class TagAliasRegistry : public ITagAliasRegistry {
12380     public:
12381         ~TagAliasRegistry() override;
12382         TagAlias const* find( std::string const& alias ) const override;
12383         std::string expandAliases( std::string const& unexpandedTestSpec ) const override;
12384         void add( std::string const& alias, std::string const& tag, SourceLineInfo const& lineInfo );
12385
12386     private:
12387         std::map<std::string, TagAlias> m_registry;
12388     };
12389
12390 } // end namespace Catch
12391
12392 // end catch_tag_alias_registry.h
12393 // start catch_startup_exception_registry.h
12394
12395 #include <vector>
12396 #include <exception>
12397
12398 namespace Catch {
12399
12400     class StartupExceptionRegistry {
12401     #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
12402     public:
12403         void add(std::exception_ptr const& exception) noexcept;
12404         std::vector<std::exception_ptr> const& getExceptions() const noexcept;
12405     private:
12406         std::vector<std::exception_ptr> m_exceptions;
12407     #endif
12408     };
12409
12410 } // end namespace Catch
12411
12412 // end catch_startup_exception_registry.h
12413 // start catch_singletons.hpp
12414
12415 namespace Catch {
12416
12417     struct ISingleton {
12418         virtual ~ISingleton();
12419     };
12420
12421     void addSingleton( ISingleton* singleton );
12422     void cleanupSingletons();
12423
12424     template<typename SingletonImplT, typename InterfaceT = SingletonImplT, typename MutableInterfaceT
= InterfaceT>
12425     class Singleton : SingletonImplT, public ISingleton {
12426
12427     public:
12428         static auto getInternal() -> Singleton* {
12429             static Singleton* s_instance = nullptr;
12430             if( !s_instance ) {
12431                 s_instance = new Singleton;
12432                 addSingleton( s_instance );
12433             }
12434             return s_instance;
12435         }
12436
12437         static auto get() -> InterfaceT const& {
12438             return *getInternal();
12439         }
12440         static auto getMutable() -> MutableInterfaceT& {
12441             return *getInternal();
12442         }
12443     };
12444
12445 } // namespace Catch
12446
12447 // end catch_singletons.hpp
12448 namespace Catch {
12449
12450     namespace {

```

```

12451
12452     class RegistryHub : public IRegistryHub, public IMutableRegistryHub,
12453                       private NonCopyable {
12454
12455     public: // IRegistryHub
12456         RegistryHub() = default;
12457         IReporterRegistry const& getReporterRegistry() const override {
12458             return m_reporterRegistry;
12459         }
12460         ITestCaseRegistry const& getTestCaseRegistry() const override {
12461             return m_testCaseRegistry;
12462         }
12463         IExceptionTranslatorRegistry const& getExceptionTranslatorRegistry() const override {
12464             return m_exceptionTranslatorRegistry;
12465         }
12466         ITagAliasRegistry const& getTagAliasRegistry() const override {
12467             return m_tagAliasRegistry;
12468         }
12469         StartupExceptionRegistry const& getStartupExceptionRegistry() const override {
12470             return m_exceptionRegistry;
12471         }
12472
12473     public: // IMutableRegistryHub
12474         void registerReporter( std::string const& name, IReporterFactoryPtr const& factory )
12475     override {
12476         m_reporterRegistry.registerReporter( name, factory );
12477         void registerListener( IReporterFactoryPtr const& factory ) override {
12478             m_reporterRegistry.registerListener( factory );
12479         }
12480         void registerTest( TestCase const& testInfo ) override {
12481             m_testCaseRegistry.registerTest( testInfo );
12482         }
12483         void registerTranslator( const IExceptionTranslator* translator ) override {
12484             m_exceptionTranslatorRegistry.registerTranslator( translator );
12485         }
12486         void registerTagAlias( std::string const& alias, std::string const& tag, SourceLineInfo
12487     const& lineInfo ) override {
12488             m_tagAliasRegistry.add( alias, tag, lineInfo );
12489             void registerStartupException() noexcept override {
12490                 #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
12491                     m_exceptionRegistry.add(std::current_exception());
12492                 #else
12493                     CATCH_INTERNAL_ERROR("Attempted to register active exception under
12494     CATCH_CONFIG_DISABLE_EXCEPTIONS!");
12495                 #endif
12496             }
12497             IMutableEnumValuesRegistry& getMutableEnumValuesRegistry() override {
12498                 return m_enumValuesRegistry;
12499             }
12500         private:
12501             TestRegistry m_testCaseRegistry;
12502             ReporterRegistry m_reporterRegistry;
12503             ExceptionTranslatorRegistry m_exceptionTranslatorRegistry;
12504             TagAliasRegistry m_tagAliasRegistry;
12505             StartupExceptionRegistry m_exceptionRegistry;
12506             Detail::EnumValuesRegistry m_enumValuesRegistry;
12507         };
12508     };
12509
12510     using RegistryHubSingleton = Singleton<RegistryHub, IRegistryHub, IMutableRegistryHub>;
12511
12512     IRegistryHub const& getRegistryHub() {
12513         return RegistryHubSingleton::get();
12514     }
12515     IMutableRegistryHub& getMutableRegistryHub() {
12516         return RegistryHubSingleton::getMutable();
12517     }
12518     void cleanUp() {
12519         cleanupSingletons();
12520         cleanUpContext();
12521     }
12522     std::string translateActiveException() {
12523         return getRegistryHub().getExceptionTranslatorRegistry().translateActiveException();
12524     }
12525
12526 } // end namespace Catch
12527 // end catch_registry_hub.cpp
12528 // start catch_reporter_registry.cpp
12529
12530 namespace Catch {
12531
12532     ReporterRegistry::~ReporterRegistry() = default;
12533
12534     IStreamingReporterPtr ReporterRegistry::create( std::string const& name, IConfigPtr const& config

```

```

    ) const {
12535         auto it = m_factories.find( name );
12536         if( it == m_factories.end() )
12537             return nullptr;
12538         return it->second->create( ReporterConfig( config ) );
12539     }
12540
12541     void ReporterRegistry::registerReporter( std::string const& name, IReporterFactoryPtr const&
factory ) {
12542         m_factories.emplace(name, factory);
12543     }
12544     void ReporterRegistry::registerListener( IReporterFactoryPtr const& factory ) {
12545         m_listeners.push_back( factory );
12546     }
12547
12548     IReporterRegistry::FactoryMap const& ReporterRegistry::getFactories() const {
12549         return m_factories;
12550     }
12551     IReporterRegistry::Listeners const& ReporterRegistry::getListeners() const {
12552         return m_listeners;
12553     }
12554
12555 }
12556 // end catch_reporter_registry.cpp
12557 // start catch_result_type.cpp
12558
12559 namespace Catch {
12560
12561     boolisOk( ResultWas::OfType resultType ) {
12562         return ( resultType & ResultWas::FailureBit ) == 0;
12563     }
12564     bool isJustInfo( int flags ) {
12565         return flags == ResultWas::Info;
12566     }
12567
12568     ResultDisposition::Flags operator | ( ResultDisposition::Flags lhs, ResultDisposition::Flags rhs )
{
12569         return static_cast<ResultDisposition::Flags>( static_cast<int>( lhs ) | static_cast<int>( rhs
) );
12570     }
12571
12572     bool shouldContinueOnFailure( int flags ) { return ( flags &
ResultDisposition::ContinueOnFailure ) != 0; }
12573     bool shouldSuppressFailure( int flags ) { return ( flags & ResultDisposition::SuppressFail )
!= 0; }
12574
12575 } // end namespace Catch
12576 // end catch_result_type.cpp
12577 // start catch_run_context.cpp
12578
12579 #include <cassert>
12580 #include <algorithm>
12581 #include <sstream>
12582
12583 namespace Catch {
12584
12585     namespace Generators {
12586         struct GeneratorTracker : TestCaseTracking::TrackerBase, IGeneratorTracker {
12587             GeneratorBasePtr m_generator;
12588
12589             GeneratorTracker( TestCaseTracking::NameAndLocation const& nameAndLocation,
TrackerContext& ctx, ITracker* parent )
: TrackerBase( nameAndLocation, ctx, parent )
12590             {}
12591             ~GeneratorTracker();
12592
12593             static GeneratorTracker& acquire( TrackerContext& ctx, TestCaseTracking::NameAndLocation
const& nameAndLocation ) {
12594                 std::shared_ptr<GeneratorTracker> tracker;
12595
12596                 ITracker& currentTracker = ctx.currentTracker();
12597                 // Under specific circumstances, the generator we want
12598                 // to acquire is also the current tracker. If this is
12599                 // the case, we have to avoid looking through current
12600                 // tracker's children, and instead return the current
12601                 // tracker.
12602                 // A case where this check is important is e.g.
12603                 // for (int i = 0; i < 5; ++i) {
12604                 //     int n = GENERATE(1, 2);
12605                 // }
12606                 // without it, the code above creates 5 nested generators.
12607                 if (currentTracker.nameAndLocation() == nameAndLocation) {
12608                     auto thisTracker = currentTracker.parent().findChild(nameAndLocation);
12609                     assert(thisTracker);
12610                     assert(thisTracker->isGeneratorTracker());
12611                     tracker = std::static_pointer_cast<GeneratorTracker>(thisTracker);
12612                 }
12613             }

```

```

12614         } else if ( TestCaseTracking::ITrackerPtr childTracker = currentTracker.findChild(
nameAndLocation ) ) {
12615             assert( childTracker );
12616             assert( childTracker->isGeneratorTracker() );
12617             tracker = std::static_pointer_cast<GeneratorTracker>( childTracker );
12618         } else {
12619             tracker = std::make_shared<GeneratorTracker>( nameAndLocation, ctx,
&currentTracker );
12620             currentTracker.addChild( tracker );
12621         }
12622
12623         if( !tracker->isComplete() ) {
12624             tracker->open();
12625         }
12626
12627         return *tracker;
12628     }
12629
12630     // TrackerBase interface
12631     bool isGeneratorTracker() const override { return true; }
12632     auto hasGenerator() const -> bool override {
12633         return !!m_generator;
12634     }
12635     void close() override {
12636         TrackerBase::close();
12637         // If a generator has a child (it is followed by a section)
12638         // and none of its children have started, then we must wait
12639         // until later to start consuming its values.
12640         // This catches cases where 'GENERATE' is placed between two
12641         // 'SECTION's.
12642         // **The check for m_children.empty cannot be removed**.
12643         // doing so would break 'GENERATE' _not_ followed by 'SECTION's.
12644         const bool should_wait_for_child = [&]() {
12645             // No children -> nobody to wait for
12646             if ( m_children.empty() ) {
12647                 return false;
12648             }
12649             // If at least one child started executing, don't wait
12650             if ( std::find_if(
12651                 m_children.begin(),
12652                 m_children.end(),
12653                 []( TestCaseTracking::ITrackerPtr tracker ) {
12654                     return tracker->hasStarted();
12655                 } ) != m_children.end() ) {
12656                 return false;
12657             }
12658
12659             // No children have started. We need to check if they _can_
12660             // start, and thus we should wait for them, or they cannot
12661             // start (due to filters), and we shouldn't wait for them
12662             auto* parent = m_parent;
12663             // This is safe: there is always at least one section
12664             // tracker in a test case tracking tree
12665             while ( !parent->isSectionTracker() ) {
12666                 parent = &( parent->parent() );
12667             }
12668             assert( parent &&
12669                 "Missing root (test case) level section" );
12670
12671             auto const& parentSection =
12672                 static_cast<SectionTracker&>( *parent );
12673             auto const& filters = parentSection.getFilters();
12674             // No filters -> no restrictions on running sections
12675             if ( filters.empty() ) {
12676                 return true;
12677             }
12678
12679             for ( auto const& child : m_children ) {
12680                 if ( child->isSectionTracker() &&
12681                     std::find( filters.begin(),
12682                         filters.end(),
12683                         static_cast<SectionTracker&>( *child )
12684                             .trimmedName() ) !=
12685                         filters.end() ) {
12686                     return true;
12687                 }
12688             }
12689             return false;
12690         }();
12691
12692         // This check is a bit tricky, because m_generator->next()
12693         // has a side-effect, where it consumes generator's current
12694         // value, but we do not want to invoke the side-effect if
12695         // this generator is still waiting for any child to start.
12696         if ( should_wait_for_child ||
12697             ( m_runState == CompletedSuccessfully &&
12698               m_generator->next() ) ) {

```

```

12699         m_children.clear();
12700         m_runState = Executing;
12701     }
12702 }
12703
12704 // IGeneratorTracker interface
12705 auto getGenerator() const -> GeneratorBasePtr const& override {
12706     return m_generator;
12707 }
12708 void setGenerator( GeneratorBasePtr&& generator ) override {
12709     m_generator = std::move( generator );
12710 }
12711 };
12712 GeneratorTracker::~GeneratorTracker() {}
12713 }
12714
12715 RunContext::RunContext( IConfigPtr const& _config, IStreamingReporterPtr&& reporter)
12716 :   m_runInfo(_config->name()),
12717   m_context(getCurrentMutableContext()),
12718   m_config(_config),
12719   m_reporter(std::move(reporter)),
12720   m_lastAssertionInfo{ StringRef(), SourceLineInfo("",0), StringRef(), ResultDisposition::Normal
12721 },
12722   m_includeSuccessfulResults( m_config->includeSuccessfulResults() ||
m_reporter->getPreferences().shouldReportAllAssertions )
12723 {
12724     m_context.setRunner(this);
12725     m_context.setConfig(m_config);
12726     m_context.setResultCapture(this);
12727     m_reporter->testRunStarting(m_runInfo);
12728 }
12729 RunContext::~RunContext() {
12730     m_reporter->testRunEnded(TestRunStats(m_runInfo, m_totals, aborting()));
12731 }
12732
12733 void RunContext::testGroupStarting(std::string const& testSpec, std::size_t groupIndex,
std::size_t groupsCount) {
12734     m_reporter->testGroupStarting(GroupInfo(testSpec, groupIndex, groupsCount));
12735 }
12736
12737 void RunContext::testGroupEnded(std::string const& testSpec, Totals const& totals, std::size_t
groupIndex, std::size_t groupsCount) {
12738     m_reporter->testGroupEnded(TestGroupStats(GroupInfo(testSpec, groupIndex, groupsCount),
totals, aborting()));
12739 }
12740
12741 Totals RunContext::runTest(TestCase const& testCase) {
12742     Totals prevTotals = m_totals;
12743
12744     std::string redirectedCout;
12745     std::string redirectedCerr;
12746
12747     auto const& testInfo = testCase.getTestCaseInfo();
12748
12749     m_reporter->testCaseStarting(testInfo);
12750
12751     m_activeTestCase = &testCase;
12752
12753     ITracker& rootTracker = m_trackerContext.startRun();
12754     assert(rootTracker.isSectionTracker());
12755     static_cast<SectionTracker&>(rootTracker).addInitialFilters(m_config->getSectionsToRun());
12756     do {
12757         m_trackerContext.startCycle();
12758         m_testCaseTracker = &SectionTracker::acquire(m_trackerContext,
TestCasesTracker::NameAndLocation(testInfo.name, testInfo.lineInfo));
12759         runCurrentTest(redirectedCout, redirectedCerr);
12760     } while (!m_testCaseTracker->isSuccessfullyCompleted() && !aborting());
12761
12762     Totals deltaTotals = m_totals.delta(prevTotals);
12763     if (testInfo.expectedToFail() && deltaTotals.testCases.passed > 0) {
12764         deltaTotals.assertions.failed++;
12765         deltaTotals.testCases.passed--;
12766         deltaTotals.testCases.failed++;
12767     }
12768     m_totals.testCases += deltaTotals.testCases;
12769     m_reporter->testCaseEnded(TestCaseStats(testInfo,
deltaTotals,
redirectedCout,
redirectedCerr,
aborting()));
12770
12771     m_activeTestCase = nullptr;
12772     m_testCaseTracker = nullptr;
12773
12774     return deltaTotals;
12775 }
12776
12777 return deltaTotals;
12778 }
12779

```

```

12780
12781     IConfigPtr RunContext::config() const {
12782         return m_config;
12783     }
12784
12785     IStreamingReporter& RunContext::reporter() const {
12786         return *m_reporter;
12787     }
12788
12789     void RunContext::assertionEnded(AssertionResult const & result) {
12790         if (result.getResultType() == ResultWas::Ok) {
12791             m_totals.assertions.passed++;
12792             m_lastAssertionPassed = true;
12793         } else if (!result.isOk()) {
12794             m_lastAssertionPassed = false;
12795             if( m_activeTestCase->getTestCaseInfo().okToFail() )
12796                 m_totals.assertions.failedButOk++;
12797             else
12798                 m_totals.assertions.failed++;
12799         }
12800         else {
12801             m_lastAssertionPassed = true;
12802         }
12803
12804         // We have no use for the return value (whether messages should be cleared), because messages
12805         // were made scoped
12806         // and should be let to clear themselves out.
12807         static_cast<void>(m_reporter->assertionEnded(AssertionStats(result, m_messages, m_totals)));
12808
12809         if (result.getResultType() != ResultWas::Warning)
12810             m_messageScopes.clear();
12811
12812         // Reset working state
12813         resetAssertionInfo();
12814         m_lastResult = result;
12815     }
12816     void RunContext::resetAssertionInfo() {
12817         m_lastAssertionInfo.macroName = StringRef();
12818         m_lastAssertionInfo.capturedExpression = "{Unknown expression after the reported line}"_sr;
12819     }
12820     bool RunContext::sectionStarted(SectionInfo const & sectionInfo, Counts & assertions) {
12821         ITracker& sectionTracker = SectionTracker::acquire(m_trackerContext,
12822             TestCaseTracking::NameAndLocation(sectionInfo.name, sectionInfo.lineInfo));
12823         if (!sectionTracker.isOpen())
12824             return false;
12825         m_activeSections.push_back(&sectionTracker);
12826
12827         m_lastAssertionInfo.lineInfo = sectionInfo.lineInfo;
12828
12829         m_reporter->sectionStarting(sectionInfo);
12830
12831         assertions = m_totals.assertions;
12832
12833         return true;
12834     }
12835     auto RunContext::acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo
12836 ) -> IGeneratorTracker& {
12837         using namespace Generators;
12838         GeneratorTracker& tracker = GeneratorTracker::acquire(m_trackerContext,
12839             TestCaseTracking::NameAndLocation(
12840             static_cast<std::string>(generatorName), lineInfo ) );
12841         m_lastAssertionInfo.lineInfo = lineInfo;
12842         return tracker;
12843     }
12844     bool RunContext::testForMissingAssertions(Counts& assertions) {
12845         if (assertions.total() != 0)
12846             return false;
12847         if (!m_config->warnAboutMissingAssertions())
12848             return false;
12849         if (m_trackerContext.currentTracker().hasChildren())
12850             return false;
12851         m_totals.assertions.failed++;
12852         assertions.failed++;
12853         return true;
12854     }
12855     void RunContext::sectionEnded(SectionEndInfo const & endInfo) {
12856         Counts assertions = m_totals.assertions - endInfo.prevAssertions;
12857         bool missingAssertions = testForMissingAssertions(assertions);
12858
12859         if (!m_activeSections.empty()) {
12860             m_activeSections.back()->close();
12861             m_activeSections.pop_back();
12862         }

```



```

12863         m_reporter->sectionEnded(SectionStats(endInfo.sectionInfo, assertions,
12864         endInfo.durationInSeconds, missingAssertions));
12865         m_messages.clear();
12866         m_messageScopes.clear();
12867     }
12868     void RunContext::sectionEndedEarly(SectionEndInfo const & endInfo) {
12869         if (m_unfinishedSections.empty())
12870             m_activeSections.back()->fail();
12871         else
12872             m_activeSections.back()->close();
12873         m_activeSections.pop_back();
12874     }
12875     void RunContext::push_back(endInfo);
12876 }
12877
12878 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
12879 void RunContext::benchmarkPreparing(std::string const& name) {
12880     m_reporter->benchmarkPreparing(name);
12881 }
12882 void RunContext::benchmarkStarting( BenchmarkInfo const& info ) {
12883     m_reporter->benchmarkStarting( info );
12884 }
12885 void RunContext::benchmarkEnded( BenchmarkStats<> const& stats ) {
12886     m_reporter->benchmarkEnded( stats );
12887 }
12888 void RunContext::benchmarkFailed(std::string const & error) {
12889     m_reporter->benchmarkFailed(error);
12890 }
12891 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
12892
12893 void RunContext::pushScopedMessage(MessageInfo const & message) {
12894     m_messages.push_back(message);
12895 }
12896
12897 void RunContext::popScopedMessage(MessageInfo const & message) {
12898     m_messages.erase(std::remove(m_messages.begin(), m_messages.end(), message),
12899     m_messages.end());
12900 }
12901
12902 void RunContext::emplaceUnscopedMessage( MessageBuilder const& builder ) {
12903     m_messageScopes.emplace_back( builder );
12904 }
12905
12906 std::string RunContext::getCurrentTestName() const {
12907     return m_activeTestCase
12908         ? m_activeTestCase->getTestCaseInfo().name
12909         : std::string();
12910 }
12911
12912 const AssertionResult * RunContext::getLastResult() const {
12913     return &(*m_lastResult);
12914 }
12915
12916 void RunContext::exceptionEarlyReported() {
12917     m_shouldReportUnexpected = false;
12918 }
12919
12920 void RunContext::handleFatalErrorCondition( StringRef message ) {
12921     // First notify reporter that bad things happened
12922     m_reporter->fatalErrorEncountered(message);
12923
12924     // Don't rebuild the result -- the stringification itself can cause more fatal errors
12925     // Instead, fake a result data.
12926     AssertionResultData tempResult( ResultWas::FatalErrorCondition, { false } );
12927     tempResult.message = static_cast<std::string>(message);
12928     AssertionResult result(m_lastAssertionInfo, tempResult);
12929
12930     assertionEnded(result);
12931
12932     handleUnfinishedSections();
12933
12934     // Recreate section for test case (as we will lose the one that was in scope)
12935     auto const& testCaseInfo = m_activeTestCase->getTestCaseInfo();
12936     SectionInfo testCaseSection(testCaseInfo.lineInfo, testCaseInfo.name);
12937
12938     Counts assertions;
12939     assertions.failed = 1;
12940     SectionStats testCaseSectionStats(testCaseSection, assertions, 0, false);
12941     m_reporter->sectionEnded(testCaseSectionStats);
12942
12943     auto const& testInfo = m_activeTestCase->getTestInfo();
12944
12945     Totals deltaTotals;
12946     deltaTotals.testCases.failed = 1;
12947     deltaTotals.assertions.failed = 1;
12948     m_reporter->testCaseEnded( TestCaseStats(testInfo,

```

```

12948             deltaTotals,
12949             std::string(),
12950             std::string(),
12951             false));
12952     m_totals.testCases.failed++;
12953     testGroupEnded(std::string(), m_totals, 1, 1);
12954     m_reporter->testRunEnded(TestRunStats(m_runInfo, m_totals, false));
12955 }
12956
12957 bool RunContext::lastAssertionPassed() {
12958     return m_lastAssertionPassed;
12959 }
12960
12961 void RunContext::assertionPassed() {
12962     m_lastAssertionPassed = true;
12963     ++m_totals.assertions.passed;
12964     resetAssertionInfo();
12965     m_messageScopes.clear();
12966 }
12967
12968 bool RunContext::aborting() const {
12969     return m_totals.assertions.failed >= static_cast<std::size_t>(m_config->abortAfter());
12970 }
12971
12972 void RunContext::runCurrentTest(std::string & redirectedCout, std::string & redirectedCerr) {
12973     auto const& testCaseInfo = m_activeTestCase->getTestCaseInfo();
12974     SectionInfo testCaseSection(testCaseInfo.lineInfo, testCaseInfo.name);
12975     m_reporter->sectionStarting(testCaseSection);
12976     Counts prevAssertions = m_totals.assertions;
12977     double duration = 0;
12978     m_shouldReportUnexpected = true;
12979     m_lastAssertionInfo = { "TEST_CASE"_sr, testCaseInfo.lineInfo, StringRef(),
ResultDisposition::Normal };
12980
12981     seedRng(*m_config);
12982
12983     Timer timer;
12984     CATCH_TRY {
12985         if (m_reporter->getPreferences().shouldRedirectStdOut) {
12986             #if !defined(CATCH_CONFIG_EXPERIMENTAL_REDIRECT)
12987                 RedirectedStreams redirectedStreams(redirectedCout, redirectedCerr);
12988
12989                 timer.start();
12990                 invokeActiveTestCase();
12991             #else
12992                 OutputRedirect r(redirectedCout, redirectedCerr);
12993                 timer.start();
12994                 invokeActiveTestCase();
12995             #endif
12996         } else {
12997             timer.start();
12998             invokeActiveTestCase();
12999         }
13000         duration = timer.getElapsedSeconds();
13001     } CATCH_CATCH_ANON (TestFailureException&) {
13002         // This just means the test was aborted due to failure
13003     } CATCH_CATCH_ALL {
13004         // Under CATCH_CONFIG_FAST_COMPILE, unexpected exceptions under REQUIRE assertions
13005         // are reported without translation at the point of origin.
13006         if( m_shouldReportUnexpected ) {
13007             AssertionReaction dummyReaction;
13008             handleUnexpectedInflightException( m_lastAssertionInfo, translateActiveException(),
dummyReaction );
13009         }
13010     }
13011     Counts assertions = m_totals.assertions - prevAssertions;
13012     bool missingAssertions = testForMissingAssertions(assertions);
13013
13014     m_testCaseTracker->close();
13015     handleUnfinishedSections();
13016     m_messages.clear();
13017     m_messageScopes.clear();
13018
13019     SectionStats testCaseSectionStats(testCaseSection, assertions, duration, missingAssertions);
13020     m_reporter->sectionEnded(testCaseSectionStats);
13021 }
13022
13023 void RunContext::invokeActiveTestCase() {
13024     FatalConditionHandlerGuard _(&m_fatalConditionhandler);
13025     m_activeTestCase->invoke();
13026 }
13027
13028 void RunContext::handleUnfinishedSections() {
13029     // If sections ended prematurely due to an exception we stored their
13030     // infos here so we can tear them down outside the unwind process.
13031     for (auto it = m_unfinishedSections.rbegin(),
itEnd = m_unfinishedSections.rend();
13032
```

```

13033         it != itEnd;
13034         ++it)
13035         sectionEnded(*it);
13036     m_unfinishedSections.clear();
13037 }
13038
13039 void RunContext::handleExpr(
13040     AssertionInfo const& info,
13041     ITransientExpression const& expr,
13042     AssertionReaction& reaction
13043 ) {
13044     m_reporter->assertionStarting( info );
13045
13046     bool negated = isFalseTest( info.resultDisposition );
13047     bool result = expr.getResult() != negated;
13048
13049     if( result ) {
13050         if (!m_includeSuccessfulResults) {
13051             assertionPassed();
13052         }
13053     } else {
13054         reportExpr(info, ResultWas::Ok, &expr, negated);
13055     }
13056 }
13057 else {
13058     reportExpr(info, ResultWas::ExpressionFailed, &expr, negated );
13059     populateReaction( reaction );
13060 }
13061 }
13062 void RunContext::reportExpr(
13063     AssertionInfo const& info,
13064     ResultWas::OfType resultType,
13065     ITransientExpression const* expr,
13066     bool negated ) {
13067
13068     m_lastAssertionInfo = info;
13069     AssertionResultData data( resultType, LazyExpression( negated ) );
13070
13071     AssertionResult assertionResult{ info, data };
13072     assertionResult.m_resultData.lazyExpression.m_transientExpression = expr;
13073
13074     assertionEnded( assertionResult );
13075 }
13076
13077 void RunContext::handleMessage(
13078     AssertionInfo const& info,
13079     ResultWas::OfType resultType,
13080     StringRef const& message,
13081     AssertionReaction& reaction
13082 ) {
13083     m_reporter->assertionStarting( info );
13084
13085     m_lastAssertionInfo = info;
13086
13087     AssertionResultData data( resultType, LazyExpression( false ) );
13088     data.message = static_cast<std::string>(message);
13089     AssertionResult assertionResult{ m_lastAssertionInfo, data };
13090     assertionEnded( assertionResult );
13091     if( !assertionResult.isOk() )
13092         populateReaction( reaction );
13093 }
13094 void RunContext::handleUnexpectedExceptionNotThrown(
13095     AssertionInfo const& info,
13096     AssertionReaction& reaction
13097 ) {
13098     handleNonExpr(info, Catch::ResultWas::DidntThrowException, reaction);
13099 }
13100
13101 void RunContext::handleUnexpectedInflightException(
13102     AssertionInfo const& info,
13103     std::string const& message,
13104     AssertionReaction& reaction
13105 ) {
13106     m_lastAssertionInfo = info;
13107
13108     AssertionResultData data( ResultWas::ThrewException, LazyExpression( false ) );
13109     data.message = message;
13110     AssertionResult assertionResult{ info, data };
13111     assertionEnded( assertionResult );
13112     populateReaction( reaction );
13113 }
13114
13115 void RunContext::populateReaction( AssertionReaction& reaction ) {
13116     reaction.shouldDebugBreak = m_config->shouldDebugBreak();
13117     reaction.shouldThrow = aborting() || (m_lastAssertionInfo.resultDisposition &
ResultDisposition::Normal);
13118 }

```

```

13119
13120 void RunContext::handleIncomplete(
13121     AssertionInfo const& info
13122 ) {
13123     m_lastAssertionInfo = info;
13124
13125     AssertionResultData data( ResultWas::ThrewException, LazyExpression( false ) );
13126     data.message = "Exception translation was disabled by CATCH_CONFIG_FAST_COMPILE";
13127     AssertionResult assertionResult{ info, data };
13128     assertionEnded( assertionResult );
13129 }
13130 void RunContext::handleNonExpr(
13131     AssertionInfo const& info,
13132     ResultWas::OfType resultType,
13133     AssertionReaction &reaction
13134 ) {
13135     m_lastAssertionInfo = info;
13136
13137     AssertionResultData data( resultType, LazyExpression( false ) );
13138     AssertionResult assertionResult{ info, data };
13139     assertionEnded( assertionResult );
13140
13141     if( !assertionResult.isOk() )
13142         populateReaction( reaction );
13143 }
13144
13145 IResultCapture& getResultCapture() {
13146     if (auto* capture = getCurrentContext().getResultCapture())
13147         return *capture;
13148     else
13149         CATCH_INTERNAL_ERROR("No result capture instance");
13150 }
13151
13152 void seedRng(IConfig const& config) {
13153     if (config.rngSeed() != 0) {
13154         std::srand(config.rngSeed());
13155         rng().seed(config.rngSeed());
13156     }
13157 }
13158
13159 unsigned int rngSeed() {
13160     return getCurrentContext().getConfig()->rngSeed();
13161 }
13162 }
13163 // end catch_run_context.cpp
13164 // start catch_section.cpp
13165
13166 namespace Catch {
13167
13168     Section::Section( SectionInfo const& info )
13169     :   m_info( info ),
13170         m_sectionIncluded( getResultCapture().sectionStarted( m_info, m_assertions ) )
13171     {
13172         m_timer.start();
13173     }
13174
13175     Section::~Section() {
13176         if( m_sectionIncluded ) {
13177             SectionEndInfo endInfo{ m_info, m_assertions, m_timer.getElapsedSeconds() };
13178             if( uncaught_exceptions() )
13179                 getResultCapture().sectionEndedEarly( endInfo );
13180             else
13181                 getResultCapture().sectionEnded( endInfo );
13182         }
13183     }
13184
13185     // This indicates whether the section should be executed or not
13186     Section::operator bool() const {
13187         return m_sectionIncluded;
13188     }
13189 }
13190 // end namespace Catch
13191 // end catch_section.cpp
13192 // start catch_section_info.cpp
13193
13194 namespace Catch {
13195
13196     SectionInfo::SectionInfo
13197     (   SourceLineInfo const& _lineInfo,
13198         std::string const& _name )
13199     :   name( _name ),
13200         lineInfo( _lineInfo )
13201     {}
13202 }
13203 // end namespace Catch
13204 // end catch_section_info.cpp

```

```

13206 // start catch_session.cpp
13207
13208 // start catch_session.h
13209
13210 #include <memory>
13211
13212 namespace Catch {
13213
13214     class Session : NonCopyable {
13215     public:
13216
13217         Session();
13218         ~Session() override;
13219
13220         void showHelp() const;
13221         void libIdentify();
13222
13223         int applyCommandLine( int argc, char const * const * argv );
13224         #if defined(CATCH_CONFIG_WCHAR) && defined(_WIN32) && defined(UNICODE)
13225             int applyCommandLine( int argc, wchar_t const * const * argv );
13226         #endif
13227
13228         void useConfigData( ConfigData const& configData );
13229
13230         template<typename CharT>
13231         int run(int argc, CharT const * const argv[]) {
13232             if (m_startupExceptions)
13233                 return 1;
13234             int returnCode = applyCommandLine(argc, argv);
13235             if (returnCode == 0)
13236                 returnCode = run();
13237             return returnCode;
13238         }
13239
13240         int run();
13241
13242         clara::Parser const& cli() const;
13243         void cli( clara::Parser const& newParser );
13244         ConfigData& configData();
13245         Config& config();
13246     private:
13247         int runInternal();
13248
13249         clara::Parser m_cli;
13250         ConfigData m_configData;
13251         std::shared_ptr<Config> m_config;
13252         bool m_startupExceptions = false;
13253     };
13254
13255 } // end namespace Catch
13256
13257 // end catch_session.h
13258 // start catch_version.h
13259
13260 #include <iosfwd>
13261
13262 namespace Catch {
13263
13264     // Versioning information
13265     struct Version {
13266         Version( Version const& ) = delete;
13267         Version& operator=( Version const& ) = delete;
13268         Version( unsigned int _majorVersion,
13269                 unsigned int _minorVersion,
13270                 unsigned int _patchNumber,
13271                 char const * _branchName,
13272                 unsigned int _buildNumber );
13273
13274         unsigned int const majorVersion;
13275         unsigned int const minorVersion;
13276         unsigned int const patchNumber;
13277
13278         // buildNumber is only used if branchName is not null
13279         char const * const branchName;
13280         unsigned int const buildNumber;
13281
13282         friend std::ostream& operator << ( std::ostream& os, Version const& version );
13283     };
13284
13285     Version const& libraryVersion();
13286 }
13287
13288 // end catch_version.h
13289 #include <cstdlib>
13290 #include <iomanip>
13291 #include <set>
13292 #include <iterator>

```

```

13293
13294 namespace Catch {
13295
13296     namespace {
13297         const int MaxExitCode = 255;
13298
13299         IStreamingReporterPtr createReporter(std::string const& reporterName, IConfigPtr const&
13300 config) {
13301             auto reporter = Catch::getRegistryHub().getReporterRegistry().create(reporterName,
13302 config);
13303             CATCH_ENFORCE(reporter, "No reporter registered with name: '" << reporterName << "'");
13304
13305             return reporter;
13306
13307         IStreamingReporterPtr makeReporter(std::shared_ptr<Config> const& config) {
13308             if (Catch::getRegistryHub().getReporterRegistry().getListeners().empty()) {
13309                 return createReporter(config->getReporterName(), config);
13310             }
13311
13312             // On older platforms, returning std::unique_ptr<ListeningReporter>
13313             // when the return type is std::unique_ptr<IStreamingReporter>
13314             // doesn't compile without a std::move call. However, this causes
13315             // a warning on newer platforms. Thus, we have to work around
13316             // it a bit and downcast the pointer manually.
13317             auto ret = std::unique_ptr<IStreamingReporter>(new ListeningReporter);
13318             auto& multi = static_cast<ListeningReporter*>(*ret);
13319             auto const& listeners = Catch::getRegistryHub().getReporterRegistry().getListeners();
13320             for (auto const& listener : listeners) {
13321                 multi.addListener(listener->create(Catch::ReporterConfig(config)));
13322             }
13323             multi.addReporter(createReporter(config->getReporterName(), config));
13324             return ret;
13325         }
13326
13327         class TestGroup {
13328         public:
13329             explicit TestGroup(std::shared_ptr<Config> const& config)
13330             : m_config{config}
13331             , m_context{config, makeReporter(config)}
13332             {
13333                 auto const& allTestCases = getAllTestCasesSorted(*m_config);
13334                 m_matches = m_config->testSpec().matchesByFilter(allTestCases, *m_config);
13335                 auto const& invalidArgs = m_config->testSpec().getInvalidArgs();
13336
13337                 if (m_matches.empty() && invalidArgs.empty()) {
13338                     for (auto const& test : allTestCases)
13339                         if (!test.isHidden())
13340                             m_tests.emplace(&test);
13341                 } else {
13342                     for (auto const& match : m_matches)
13343                         m_tests.insert(match.tests.begin(), match.tests.end());
13344                 }
13345             }
13346
13347             Totals execute() {
13348                 auto const& invalidArgs = m_config->testSpec().getInvalidArgs();
13349                 Totals totals;
13350                 m_context.testGroupStarting(m_config->name(), 1, 1);
13351                 for (auto const& testCase : m_tests) {
13352                     if (!m_context.aborting())
13353                         totals += m_context.runTest(*testCase);
13354                     else
13355                         m_context.reporter().skipTest(*testCase);
13356                 }
13357
13358                 for (auto const& match : m_matches) {
13359                     if (match.tests.empty()) {
13360                         m_context.reporter().noMatchingTestCases(match.name);
13361                         totals.error = -1;
13362                     }
13363                 }
13364
13365                 if (!invalidArgs.empty()) {
13366                     for (auto const& invalidArg : invalidArgs)
13367                         m_context.reporter().reportInvalidArguments(invalidArg);
13368                 }
13369
13370                 m_context.testGroupEnded(m_config->name(), totals, 1, 1);
13371                 return totals;
13372             }
13373
13374         private:
13375             using Tests = std::set<TestCase const*>;
13376
13377             std::shared_ptr<Config> m_config;
13378             RunContext m_context;

```

```

13378         Tests m_tests;
13379         TestSpec::Matches m_matches;
13380     };
13381
13382     void applyFileNamesAsTags(Catch::IConfig const& config) {
13383         auto& tests = const_cast<std::vector<TestCase>&>(getAllTestCasesSorted(config));
13384         for (auto& testCase : tests) {
13385             auto tags = testCase.tags;
13386
13387             std::string filename = testCase.lineInfo.file;
13388             auto lastSlash = filename.find_last_of("\\/");
13389             if (lastSlash != std::string::npos) {
13390                 filename.erase(0, lastSlash);
13391                 filename[0] = '#';
13392             }
13393             else
13394             {
13395                 filename.insert(0, "#");
13396             }
13397
13398             auto lastDot = filename.find_last_of('.');
13399             if (lastDot != std::string::npos) {
13400                 filename.erase(lastDot);
13401             }
13402
13403             tags.push_back(std::move(filename));
13404             setTags(testCase, tags);
13405         }
13406     }
13407
13408 } // anon namespace
13409
13410 Session::Session() {
13411     static bool alreadyInstantiated = false;
13412     if( alreadyInstantiated ) {
13413         CATCH_TRY { CATCH_INTERNAL_ERROR( "Only one instance of Catch::Session can ever be used"
13414 ); }
13415         CATCH_CATCH_ALL { getMutableRegistryHub().registerStartupException(); }
13416     }
13417
13418     // There cannot be exceptions at startup in no-exception mode.
13419     #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
13419     const auto& exceptions = getRegistryHub().getStartupExceptionRegistry().getExceptions();
13420     if ( !exceptions.empty() ) {
13421         config();
13422         getCurrentMutableContext().setConfig(m_config);
13423
13424         m_startupExceptions = true;
13425         Colour colourGuard( Colour::Red );
13426         Catch::cerr() << "Errors occurred during startup!" << '\n';
13427         // iterate over all exceptions and notify user
13428         for ( const auto& ex_ptr : exceptions ) {
13429             try {
13430                 std::rethrow_exception(ex_ptr);
13431             } catch ( std::exception const& ex ) {
13432                 Catch::cerr() << Column( ex.what() ).indent(2) << '\n';
13433             }
13434         }
13435     }
13436 #endif
13437
13438     alreadyInstantiated = true;
13439     m_cli = makeCommandLineParser( m_configData );
13440 }
13441 Session::~Session() {
13442     Catch::cleanUp();
13443 }
13444
13445 void Session::showHelp() const {
13446     Catch::cout()
13447         << "\nCatch v" << libraryVersion() << "\n"
13448         << m_cli << std::endl
13449         << "For more detailed usage please see the project docs\n" << std::endl;
13450 }
13451 void Session::libIdentify() {
13452     Catch::cout()
13453         << std::left << std::setw(16) << "description: " << "A Catch2 test executable\n"
13454         << std::left << std::setw(16) << "category: " << "testframework\n"
13455         << std::left << std::setw(16) << "framework: " << "Catch Test\n"
13456         << std::left << std::setw(16) << "version: " << libraryVersion() << std::endl;
13457 }
13458
13459 int Session::applyCommandLine( int argc, char const * const * argv ) {
13460     if( m_startupExceptions )
13461         return 1;
13462
13463     auto result = m_cli.parse( clara::Args( argc, argv ) );

```

```

13464         if( !result ) {
13465             config();
13466             getCurrentMutableContext().setConfig(m_config);
13467             Catch::cerr()
13468                 << Colour( Colour::Red )
13469                 << "\nError(s) in input:\n"
13470                 << Column( result.errorMessage() ).indent( 2 )
13471                 << "\n\n";
13472             Catch::cerr() << "Run with -? for usage\n" << std::endl;
13473             return MaxExitCode;
13474         }
13475
13476         if( m_configData.showHelp )
13477             showHelp();
13478         if( m_configData.libIdentify )
13479             libIdentify();
13480         m_config.reset();
13481         return 0;
13482     }
13483
13484 #if defined(CATCH_CONFIG_WCHAR) && defined(_WIN32) && defined(UNICODE)
13485 int Session::applyCommandLine( int argc, wchar_t const * const * argv ) {
13486
13487     char **utf8Argv = new char * [ argc ];
13488
13489     for ( int i = 0; i < argc; ++i ) {
13490         int bufSize = WideCharToMultiByte( CP_UTF8, 0, argv[i], -1, nullptr, 0, nullptr, nullptr
13491 );
13492         utf8Argv[ i ] = new char [ bufSize ];
13493         WideCharToMultiByte( CP_UTF8, 0, argv[i], -1, utf8Argv[i], bufSize, nullptr, nullptr );
13494     }
13495
13496     int returnCode = applyCommandLine( argc, utf8Argv );
13497
13498     for ( int i = 0; i < argc; ++i )
13499         delete [] utf8Argv[ i ];
13500
13501     delete [] utf8Argv;
13502
13503     return returnCode;
13504 }
13505 #endif
13506
13507 void Session::useConfigData( ConfigData const& configData ) {
13508     m_configData = configData;
13509     m_config.reset();
13510 }
13511
13512 int Session::run() {
13513     if( ( m_configData.waitForKeypress & WaitForKeypress::BeforeStart ) != 0 ) {
13514         Catch::cout() << "...waiting for enter/ return before starting" << std::endl;
13515         static_cast<void>(std::getchar());
13516     }
13517     int exitCode = runInternal();
13518     if( ( m_configData.waitForKeypress & WaitForKeypress::BeforeExit ) != 0 ) {
13519         Catch::cout() << "...waiting for enter/ return before exiting, with code: " << exitCode <<
13520 std::endl;
13521         static_cast<void>(std::getchar());
13522     }
13523     return exitCode;
13524 }
13525
13526 clara::Parser const& Session::cli() const {
13527     return m_cli;
13528 }
13529 void Session::cli( clara::Parser const& newParser ) {
13530     m_cli = newParser;
13531 }
13532 ConfigData& Session::configData() {
13533     return m_configData;
13534 }
13535 Config& Session::config() {
13536     if( !m_config )
13537         m_config = std::make_shared<Config>( m_configData );
13538     return *m_config;
13539 }
13540
13541 int Session::runInternal() {
13542     if( m_startupExceptions )
13543         return 1;
13544
13545     if (m_configData.showHelp || m_configData.libIdentify) {
13546         return 0;
13547     }
13548

```



```

13549     CATCH_TRY {
13550         config(); // Force config to be constructed
13551
13552         seedRng( *m_config );
13553
13554         if( m_configData_filenamesAsTags )
13555             applyFilenamesAsTags( *m_config );
13556
13557         // Handle list request
13558         if( Option<std::size_t> listed = list( m_config ) )
13559             return (std::min)(MaxExitCode, static_cast<int>(*listed));
13560
13561         TestGroup tests { m_config };
13562         auto const totals = tests.execute();
13563
13564         if( m_config->warnAboutNoTests() && totals.error == -1 )
13565             return 2;
13566
13567         // Note that on unices only the lower 8 bits are usually used, clamping
13568         // the return value to 255 prevents false negative when some multiple
13569         // of 256 tests has failed
13570         return (std::min)(MaxExitCode, (std::max)(totals.error,
13571             static_cast<int>(totals.assertions.failed)));
13572     }
13573 #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
13574     catch( std::exception& ex ) {
13575         Catch::cerr() << ex.what() << std::endl;
13576         return MaxExitCode;
13577     }
13578 #endif
13579 } // end namespace Catch
13580 // end catch_session.cpp
13581 // start catch_singletons.cpp
13582 #include <vector>
13583 namespace Catch {
13584     namespace {
13585         static auto getSingletons() -> std::vector<ISingleton*>& {
13586             static std::vector<ISingleton*> g_singletons = nullptr;
13587             if( !g_singletons )
13588                 g_singletons = new std::vector<ISingleton*>();
13589             return g_singletons;
13590         }
13591     }
13592     ISingleton::~ISingleton() {}
13593     void addSingleton(ISingleton* singleton) {
13594         getSingletons()->push_back( singleton );
13595     }
13596     void cleanupSingletons() {
13597         auto& singletons = getSingletons();
13598         for( auto singleton : *singletons )
13599             delete singleton;
13600         delete singletons;
13601         singletons = nullptr;
13602     }
13603 } // namespace Catch
13604 // end catch_singletons.cpp
13605 // start catch_startup_exception_registry.cpp
13606 #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
13607 namespace Catch {
13608     void StartupExceptionRegistry::add( std::exception_ptr const& exception ) noexcept {
13609         CATCH_TRY {
13610             m_exceptions.push_back(exception);
13611         } CATCH_CATCH_ALL {
13612             // If we run out of memory during start-up there's really not a lot more we can do about
13613             it
13614             std::terminate();
13615         }
13616     }
13617     std::vector<std::exception_ptr> const& StartupExceptionRegistry::getExceptions() const noexcept {
13618         return m_exceptions;
13619     }
13620 } // end namespace Catch
13621 #endif
13622 // end catch_startup_exception_registry.cpp
13623 // start catch_stream.cpp
13624

```

```

13634 #include <cstdio>
13635 #include <iostream>
13636 #include <fstream>
13637 #include <sstream>
13638 #include <vector>
13639 #include <memory>
13640
13641 namespace Catch {
13642
13643     Catch::IStream::~~IStream() = default;
13644
13645     namespace Detail { namespace {
13646         template<typename WriterF, std::size_t bufferSize=256>
13647         class StreamBufImpl : public std::streambuf {
13648             char data[bufferSize];
13649             WriterF m_writer;
13650
13651         public:
13652             StreamBufImpl() {
13653                 setp( data, data + sizeof(data) );
13654             }
13655
13656             ~StreamBufImpl() noexcept {
13657                 StreamBufImpl::sync();
13658             }
13659
13660         private:
13661             int overflow( int c ) override {
13662                 sync();
13663
13664                 if( c != EOF ) {
13665                     if( pbase() == epptr() )
13666                         m_writer( std::string( 1, static_cast<char>( c ) ) );
13667                     else
13668                         sputc( static_cast<char>( c ) );
13669                 }
13670                 return 0;
13671             }
13672
13673             int sync() override {
13674                 if( pbase() != pptr() ) {
13675                     m_writer( std::string( pbase(), static_cast<std::string::size_type>( pptr() -
13676 pbase() ) ) );
13677                     setp( pbase(), epptr() );
13678                 }
13679                 return 0;
13680             }
13681         };
13682
13683         struct OutputDebugWriter {
13684
13685             void operator()( std::string const&str ) {
13686                 writeToDebugConsole( str );
13687             }
13688         };
13689
13690         class FileStream : public IStream {
13691             mutable std::ofstream m_ofs;
13692         public:
13693             FileStream( StringRef filename ) {
13694                 m_ofs.open( filename.c_str() );
13695                 CATCH_ENFORCE( !m_ofs.fail(), "Unable to open file: '" < filename < "'" );
13696             }
13697             ~FileStream() override = default;
13698         public: // IStream
13699             std::ostream& stream() const override {
13700                 return m_ofs;
13701             }
13702         };
13703
13704         class CoutStream : public IStream {
13705             mutable std::ostream m_os;
13706         public:
13707             // Store the streambuf from cout up-front because
13708             // cout may get redirected when running tests
13709             CoutStream() : m_os( Catch::cout().rdbuf() ) {}
13710             ~CoutStream() override = default;
13711         public: // IStream
13712             std::ostream& stream() const override { return m_os; }
13713         };
13714
13715         class DebugOutStream : public IStream {

```

```

13724         std::unique_ptr<StreamBufImpl<OutputDebugWriter> m_streamBuf;
13725         mutable std::ostream m_os;
13726     public:
13727         DebugOutputStream()
13728         :   m_streamBuf( new StreamBufImpl<OutputDebugWriter>() ),
13729             m_os( m_streamBuf.get() )
13730         {}
13731
13732         ~DebugOutputStream() override = default;
13733
13734     public: // IStream
13735         std::ostream& stream() const override { return m_os; }
13736     };
13737
13738 } // namespace anon::detail
13739
13740
13741
13742 auto makeStream( StringRef const &filename ) -> IStream const* {
13743     if( filename.empty() )
13744         return new Detail::CoutStream();
13745     else if( filename[0] == '%' ) {
13746         if( filename == "%debug" )
13747             return new Detail::DebugOutputStream();
13748         else
13749             CATCH_ERROR( "Unrecognised stream: '" < filename < "'" );
13750     }
13751     else
13752         return new Detail::FileStream( filename );
13753 }
13754
13755 // This class encapsulates the idea of a pool of ostringstreams that can be reused.
13756 struct StringStreams {
13757     std::vector<std::unique_ptr<std::ostringstream> m_streams;
13758     std::vector<std::size_t> m_unused;
13759     std::ostringstream m_referenceStream; // Used for copy state/ flags from
13760
13761     auto add() -> std::size_t {
13762         if( m_unused.empty() ) {
13763             m_streams.push_back( std::unique_ptr<std::ostringstream>( new std::ostringstream ) );
13764             return m_streams.size()-1;
13765         }
13766         else {
13767             auto index = m_unused.back();
13768             m_unused.pop_back();
13769             return index;
13770         }
13771     }
13772
13773     void release( std::size_t index ) {
13774         m_streams[index]->copyfmt( m_referenceStream ); // Restore initial flags and other state
13775         m_unused.push_back(index);
13776     }
13777 };
13778
13779 ReusableStringStream::ReusableStringStream()
13780 :   m_index( Singleton<StringStreams>::getMutable().add() ),
13781     m_oss( Singleton<StringStreams>::getMutable().m_streams[m_index].get() )
13782 {}
13783
13784 ReusableStringStream::~ReusableStringStream() {
13785     static_cast<std::ostringstream*>( m_oss )->str("");
13786     m_oss->clear();
13787     Singleton<StringStreams>::getMutable().release( m_index );
13788 }
13789
13790 auto ReusableStringStream::str() const -> std::string {
13791     return static_cast<std::ostringstream*>( m_oss )->str();
13792 }
13793
13794
13795
13796 #ifndef CATCH_CONFIG_NOSTDOUT // If you #define this you must implement these functions
13797     std::ostream& cout() { return std::cout; }
13798     std::ostream& cerr() { return std::cerr; }
13799     std::ostream& clog() { return std::clog; }
13800 #endif
13801 }
13802 // end catch_stream.cpp
13803 // start catch_string_manip.cpp
13804
13805 #include <algorithm>
13806 #include <ostream>
13807 #include <cstring>
13808 #include <cctype>
13809 #include <vector>
13810
13811 namespace Catch {
13812

```

```

13813 namespace {
13814     char toLowerCh(char c) {
13815         return static_cast<char>( std::tolower( static_cast<unsigned char>(c) ) );
13816     }
13817 }
13818
13819 bool startsWith( std::string const& s, std::string const& prefix ) {
13820     return s.size() >= prefix.size() && std::equal(prefix.begin(), prefix.end(), s.begin());
13821 }
13822 bool startsWith( std::string const& s, char prefix ) {
13823     return !s.empty() && s[0] == prefix;
13824 }
13825 bool endsWith( std::string const& s, std::string const& suffix ) {
13826     return s.size() >= suffix.size() && std::equal(suffix.rbegin(), suffix.rend(), s.rbegin());
13827 }
13828 bool endsWith( std::string const& s, char suffix ) {
13829     return !s.empty() && s[s.size()-1] == suffix;
13830 }
13831 bool contains( std::string const& s, std::string const& infix ) {
13832     return s.find( infix ) != std::string::npos;
13833 }
13834 void toLowerInPlace( std::string& s ) {
13835     std::transform( s.begin(), s.end(), s.begin(), toLowerCh );
13836 }
13837 std::string toLower( std::string const& s ) {
13838     std::string lc = s;
13839     toLowerInPlace( lc );
13840     return lc;
13841 }
13842 std::string trim( std::string const& str ) {
13843     static char const* whitespaceChars = "\n\r\t ";
13844     std::string::size_type start = str.find_first_not_of( whitespaceChars );
13845     std::string::size_type end = str.find_last_not_of( whitespaceChars );
13846
13847     return start != std::string::npos ? str.substr( start, 1+end-start ) : std::string();
13848 }
13849
13850 StringRef trim(StringRef ref) {
13851     const auto is_ws = [](char c) {
13852         return c == ' ' || c == '\t' || c == '\n' || c == '\r';
13853     };
13854     size_t real_begin = 0;
13855     while (real_begin < ref.size() && is_ws(ref[real_begin])) { ++real_begin; }
13856     size_t real_end = ref.size();
13857     while (real_end > real_begin && is_ws(ref[real_end - 1])) { --real_end; }
13858
13859     return ref.substr(real_begin, real_end - real_begin);
13860 }
13861
13862 bool replaceInPlace( std::string& str, std::string const& replaceThis, std::string const& withThis
13863 ) {
13864     bool replaced = false;
13865     std::size_t i = str.find( replaceThis );
13866     while( i != std::string::npos ) {
13867         replaced = true;
13868         str = str.substr( 0, i ) + withThis + str.substr( i+replaceThis.size() );
13869         if( i < str.size()-withThis.size() )
13870             i = str.find( replaceThis, i+withThis.size() );
13871         else
13872             i = std::string::npos;
13873     }
13874     return replaced;
13875 }
13876
13877 std::vector<StringRef> splitStringRef( StringRef str, char delimiter ) {
13878     std::vector<StringRef> subStrings;
13879     std::size_t start = 0;
13880     for(std::size_t pos = 0; pos < str.size(); ++pos ) {
13881         if( str[pos] == delimiter ) {
13882             if( pos - start > 1 )
13883                 subStrings.push_back( str.substr( start, pos-start ) );
13884             start = pos+1;
13885         }
13886     }
13887     if( start < str.size() )
13888         subStrings.push_back( str.substr( start, str.size()-start ) );
13889     return subStrings;
13890 }
13891
13892 pluralise::pluralise( std::size_t count, std::string const& label )
13893 :   m_count( count ),
13894     m_label( label )
13895 {}
13896
13897 std::ostream& operator << ( std::ostream& os, pluralise const& pluraliser ) {
13898     os << pluraliser.m_count << ' ' << pluraliser.m_label;
13899     if( pluraliser.m_count != 1 )

```

```

13899         os << 's';
13900         return os;
13901     }
13902 }
13903 }
13904 // end catch_string_manip.cpp
13905 // start catch_stringref.cpp
13906
13907 #include <algorithm>
13908 #include <ostream>
13909 #include <cstring>
13910 #include <cstdint>
13911
13912 namespace Catch {
13913     StringRef::StringRef( char const* rawChars ) noexcept
13914     : StringRef( rawChars, static_cast<StringRef::size_type>(std::strlen(rawChars)) )
13915     {}
13916
13917     auto StringRef::c_str() const -> char const* {
13918         CATCH_ENFORCE(isNullTerminated(), "Called StringRef::c_str() on a non-null-terminated
instance");
13919         return m_start;
13920     }
13921     auto StringRef::data() const noexcept -> char const* {
13922         return m_start;
13923     }
13924
13925     auto StringRef::substr( size_type start, size_type size ) const noexcept -> StringRef {
13926         if (start < m_size) {
13927             return StringRef(m_start + start, (std::min)(m_size - start, size));
13928         } else {
13929             return StringRef();
13930         }
13931     }
13932     auto StringRef::operator == ( StringRef const& other ) const noexcept -> bool {
13933         return m_size == other.m_size
            && (std::memcmp( m_start, other.m_start, m_size ) == 0);
13934     }
13935
13936     auto operator << ( std::ostream& os, StringRef const& str ) -> std::ostream& {
13937         return os.write(str.data(), str.size());
13938     }
13939
13940     auto operator+=( std::string& lhs, StringRef const& rhs ) -> std::string& {
13941         lhs.append(rhs.data(), rhs.size());
13942         return lhs;
13943     }
13944 }
13945
13946 } // namespace Catch
13947 // end catch_stringref.cpp
13948 // start catch_tag_alias.cpp
13949
13950 namespace Catch {
13951     TagAlias::TagAlias(std::string const& _tag, SourceLineInfo _lineInfo): tag(_tag),
lineInfo(_lineInfo) {}
13952 }
13953 // end catch_tag_alias.cpp
13954 // start catch_tag_alias_autoregistrar.cpp
13955
13956 namespace Catch {
13957     RegistrarForTagAliases::RegistrarForTagAliases(char const* alias, char const* tag, SourceLineInfo
const& lineInfo) {
13958         CATCH_TRY {
13959             getMutableRegistryHub().registerTagAlias(alias, tag, lineInfo);
13960         } CATCH_CATCH_ALL {
13961             // Do not throw when constructing global objects, instead register the exception to be
13962             processed later
13963             getMutableRegistryHub().registerStartupException();
13964         }
13965     }
13966 }
13967 }
13968 // end catch_tag_alias_autoregistrar.cpp
13969 // start catch_tag_alias_registry.cpp
13970
13971 #include <sstream>
13972
13973 namespace Catch {
13974     TagAliasRegistry::~TagAliasRegistry() {}
13975
13976     TagAlias const* TagAliasRegistry::find( std::string const& alias ) const {
13977         auto it = m_registry.find( alias );
13978         if( it != m_registry.end() )
13979             return &(it->second);
13980         else
13981             return nullptr;

```

```

13982         return nullptr;
13983     }
13984
13985     std::string TagAliasRegistry::expandAliases( std::string const& unexpandedTestSpec ) const {
13986         std::string expandedTestSpec = unexpandedTestSpec;
13987         for( auto const& registryKvp : m_registry ) {
13988             std::size_t pos = expandedTestSpec.find( registryKvp.first );
13989             if( pos != std::string::npos ) {
13990                 expandedTestSpec = expandedTestSpec.substr( 0, pos ) +
13991                                     registryKvp.second.tag +
13992                                     expandedTestSpec.substr( pos + registryKvp.first.size() );
13993             }
13994         }
13995         return expandedTestSpec;
13996     }
13997
13998     void TagAliasRegistry::add( std::string const& alias, std::string const& tag, SourceLineInfo
13999     const& lineInfo ) {
14000         CATCH_ENFORCE( startsWith(alias, "[@]") && endsWith(alias, ']'),
14001             "error: tag alias, '" < alias < "' is not of the form [@alias name].\n" <
14002             lineInfo );
14003
14004         CATCH_ENFORCE( m_registry.insert( std::make_pair( alias, TagAlias( tag, lineInfo ) ) ).second,
14005             "error: tag alias, '" < alias < "' already registered.\n"
14006             < "\tFirst seen at: " < find( alias )->lineInfo < "\n"
14007             < "\tRedefined at: " < lineInfo );
14008     }
14009
14010     ITagAliasRegistry::~ITagAliasRegistry() {}
14011
14012     ITagAliasRegistry const& ITagAliasRegistry::get() {
14013         return getRegistryHub().getTagAliasRegistry();
14014     }
14015 } // end namespace Catch
14016 // end catch_tag_alias_registry.cpp
14017 // start catch_test_case_info.cpp
14018 #include <cctype>
14019 #include <exception>
14020 #include <algorithm>
14021 #include <sstream>
14022
14023 namespace Catch {
14024     namespace {
14025         namespace TestCaseInfo {
14026             SpecialProperties parseSpecialTag( std::string const& tag ) {
14027                 if( startsWith( tag, '.' ) ||
14028                     tag == "!hide" )
14029                     return TestCaseInfo::IsHidden;
14030                 else if( tag == "!throws" )
14031                     return TestCaseInfo::Throws;
14032                 else if( tag == "!shouldfail" )
14033                     return TestCaseInfo::ShouldFail;
14034                 else if( tag == "!mayfail" )
14035                     return TestCaseInfo::MayFail;
14036                 else if( tag == "!nonportable" )
14037                     return TestCaseInfo::NonPortable;
14038                 else if( tag == "!benchmark" )
14039                     return static_cast<TestCaseInfo::SpecialProperties>( TestCaseInfo::Benchmark |
14040                         TestCaseInfo::IsHidden );
14041                 else
14042                     return TestCaseInfo::None;
14043             }
14044             bool isReservedTag( std::string const& tag ) {
14045                 return parseSpecialTag( tag ) == TestCaseInfo::None && tag.size() > 0 && !std::isalnum(
14046                     static_cast<unsigned char>(tag[0]) );
14047             }
14048             void enforceNotReservedTag( std::string const& tag, SourceLineInfo const& _lineInfo ) {
14049                 CATCH_ENFORCE( !isReservedTag(tag),
14050                     "Tag name: [" < tag < "] is not allowed.\n"
14051                     < "Tag names starting with non alphanumeric characters are reserved\n"
14052                     < _lineInfo );
14053             }
14054         }
14055         TestCase makeTestCase( ITestInvoker* _testCase,
14056                             std::string const& _className,
14057                             NameAndTags const& nameAndTags,
14058                             SourceLineInfo const& _lineInfo )
14059         {
14060             bool isHidden = false;
14061
14062             // Parse out tags
14063             std::vector<std::string> tags;
14064             std::string desc, tag;
14065             bool inTag = false;

```

```

14065         for (char c : nameAndTags.tags) {
14066             if ( !inTag ) {
14067                 if( c == '[' )
14068                     inTag = true;
14069                 else
14070                     desc += c;
14071             }
14072             else {
14073                 if( c == ']' ) {
14074                     TestCaseInfo::SpecialProperties prop = parseSpecialTag( tag );
14075                     if ( ( prop & TestCaseInfo::IsHidden ) != 0 )
14076                         isHidden = true;
14077                     else if( prop == TestCaseInfo::None )
14078                         enforceNotReservedTag( tag, _lineInfo );
14079
14080                     // Merged hide tags like `[.approvals]` should be added as
14081                     // `[.]approvals`. The `[.]` is added at later point, so
14082                     // we only strip the prefix
14083                     if (startsWith(tag, '.') && tag.size() > 1) {
14084                         tag.erase(0, 1);
14085                     }
14086                     tags.push_back( tag );
14087                     tag.clear();
14088                     inTag = false;
14089                 }
14090                 else
14091                     tag += c;
14092             }
14093         }
14094         if( isHidden ) {
14095             // Add all "hidden" tags to make them behave identically
14096             tags.insert( tags.end(), { ".", "hide" } );
14097         }
14098
14099         TestCaseInfo info( static_cast<std::string>(nameAndTags.name), _className, desc, tags,
14100 _lineInfo );
14101         return TestCase( _testCase, std::move(info) );
14102     }
14103
14104     void setTags( TestCaseInfo& testCaseInfo, std::vector<std::string> tags ) {
14105         std::sort(begin(tags), end(tags));
14106         tags.erase(std::unique(begin(tags), end(tags)), end(tags));
14107         testCaseInfo.lcaseTags.clear();
14108
14109         for( auto const& tag : tags ) {
14110             std::string lcaseTag = toLower( tag );
14111             testCaseInfo.properties = static_cast<TestCaseInfo::SpecialProperties>(
14112                 testCaseInfo.properties | parseSpecialTag( lcaseTag ) );
14113             testCaseInfo.lcaseTags.push_back( lcaseTag );
14114         }
14115         testCaseInfo.tags = std::move(tags);
14116     }
14117
14118     TestCaseInfo::TestCaseInfo( std::string const& _name,
14119                                 std::string const& _className,
14120                                 std::string const& _description,
14121                                 std::vector<std::string> const& _tags,
14122                                 SourceLineInfo const& _lineInfo )
14123     :   name( _name ),
14124         className( _className ),
14125         description( _description ),
14126         lineInfo( _lineInfo ),
14127         properties( None )
14128     {
14129         setTags( *this, _tags );
14130     }
14131
14132     bool TestCaseInfo::isHidden() const {
14133         return ( properties & IsHidden ) != 0;
14134     }
14135
14136     bool TestCaseInfo::throws() const {
14137         return ( properties & Throws ) != 0;
14138     }
14139
14140     bool TestCaseInfo::okToFail() const {
14141         return ( properties & (ShouldFail | MayFail) ) != 0;
14142     }
14143
14144     bool TestCaseInfo::expectedToFail() const {
14145         return ( properties & ShouldFail ) != 0;
14146     }
14147
14148     std::string TestCaseInfo::tagsAsString() const {
14149         std::string ret;
14150         // '[' and ']' per tag
14151         std::size_t full_size = 2 * tags.size();
14152         for (const auto& tag : tags) {
14153             full_size += tag.size();
14154         }
14155     }

```

```

14150         ret.reserve(full_size);
14151         for (const auto& tag : tags) {
14152             ret.push_back('[');
14153             ret.append(tag);
14154             ret.push_back(']');
14155         }
14156         return ret;
14157     }
14158 }, test( testCase ) {}
14159
14160     TestCase::TestCase( ITestInvoker* testCase, TestCaseInfo&& info ) : TestCaseInfo( std::move(info)
14161 ), test( testCase ) {}
14162
14163     TestCase TestCase::withName( std::string const& _newName ) const {
14164         TestCase other( *this );
14165         other.name = _newName;
14166         return other;
14167     }
14168
14169     void TestCase::invoke() const {
14170         test->invoke();
14171     }
14172
14173     bool TestCase::operator == ( TestCase const& other ) const {
14174         return test.get() == other.test.get() &&
14175             name == other.name &&
14176             className == other.className;
14177     }
14178
14179     bool TestCase::operator < ( TestCase const& other ) const {
14180         return name < other.name;
14181     }
14182
14183     TestCaseInfo const& TestCase::getTestCaseInfo() const
14184     {
14185         return *this;
14186     }
14187 } // end namespace Catch
14188 // end catch_test_case_info.cpp
14189 // start catch_test_case_registry_impl.cpp
14190
14191 #include <algorithm>
14192 #include <sstream>
14193
14194 namespace Catch {
14195     namespace {
14196         struct TestHasher {
14197             using hash_t = uint64_t;
14198
14199             explicit TestHasher( hash_t hashSuffix ) :
14200                 m_hashSuffix{ hashSuffix } {}
14201
14202             uint32_t operator()( TestCase const& t ) const {
14203                 // FNV-1a hash with multiplication fold.
14204                 const hash_t prime = 1099511628211u;
14205                 hash_t hash = 14695981039346656037u;
14206                 for ( const char c : t.name ) {
14207                     hash ^= c;
14208                     hash *= prime;
14209                 }
14210                 hash ^= m_hashSuffix;
14211                 hash *= prime;
14212                 const uint32_t low{ static_cast<uint32_t>( hash ) };
14213                 const uint32_t high{ static_cast<uint32_t>( hash >> 32 ) };
14214                 return low * high;
14215             }
14216
14217             private:
14218                 hash_t m_hashSuffix;
14219         };
14220     }
14221 } // end unnamed namespace
14222
14223     std::vector<TestCase> sortTests( IConfig const& config, std::vector<TestCase> const&
14224     unsortedTestCases ) {
14225         switch( config.runOrder() ) {
14226             case RunTests::InDeclarationOrder:
14227                 // already in declaration order
14228                 break;
14229             case RunTests::InLexicographicalOrder: {
14230                 std::vector<TestCase> sorted = unsortedTestCases;
14231                 std::sort( sorted.begin(), sorted.end() );
14232                 return sorted;
14233             }
14234         }

```



```

14235         case RunTests::InRandomOrder: {
14236             seedRng( config );
14237             TestHasher h{ config.rngSeed() };
14238
14239             using hashedTest = std::pair<TestHasher::hash_t, TestCase const*>;
14240             std::vector<hashedTest> indexed_tests;
14241             indexed_tests.reserve( unsortedTestCases.size() );
14242
14243             for (auto const& testCase : unsortedTestCases) {
14244                 indexed_tests.emplace_back(h(testCase), &testCase);
14245             }
14246
14247             std::sort(indexed_tests.begin(), indexed_tests.end(),
14248                 [](hashedTest const& lhs, hashedTest const& rhs) {
14249                     if (lhs.first == rhs.first) {
14250                         return lhs.second->name < rhs.second->name;
14251                     }
14252                     return lhs.first < rhs.first;
14253                 });
14254
14255             std::vector<TestCase> sorted;
14256             sorted.reserve( indexed_tests.size() );
14257
14258             for (auto const& hashed : indexed_tests) {
14259                 sorted.emplace_back(*hashed.second);
14260             }
14261
14262             return sorted;
14263         }
14264     }
14265     return unsortedTestCases;
14266 }
14267
14268 bool isThrowSafe( TestCase const& testCase, IConfig const& config ) {
14269     return !testCasethrows() || config.allowThrows();
14270 }
14271
14272 bool matchTest( TestCase const& testCase, TestSpec const& testSpec, IConfig const& config ) {
14273     return testSpec.matches( testCase ) && isThrowSafe( testCase, config );
14274 }
14275
14276 void enforceNoDuplicateTestCases( std::vector<TestCase> const& functions ) {
14277     std::set<TestCase> seenFunctions;
14278     for( auto const& function : functions ) {
14279         auto prev = seenFunctions.insert( function );
14280         CATCH_ENFORCE( prev.second,
14281             "error: TEST_CASE( \"" << function.name << "\"" ) already defined.\n"
14282             << "\tFirst seen at " << prev.first->getTestCaseInfo().lineInfo << "\n"
14283             << "\tRedefined at " << function.getTestCaseInfo().lineInfo );
14284     }
14285 }
14286
14287 std::vector<TestCase> filterTests( std::vector<TestCase> const& testCases, TestSpec const&
testSpec, IConfig const& config ) {
14288     std::vector<TestCase> filtered;
14289     filtered.reserve( testCases.size() );
14290     for (auto const& testCase : testCases) {
14291         if ((!testSpec.hasFilters() && !testCase.isHidden()) ||
14292             (testSpec.hasFilters() && matchTest(testCase, testSpec, config))) {
14293             filtered.push_back(testCase);
14294         }
14295     }
14296     return filtered;
14297 }
14298 std::vector<TestCase> const& getAllTestCasesSorted( IConfig const& config ) {
14299     return getRegistryHub().getTestCaseRegistry().getAllTestsSorted( config );
14300 }
14301
14302 void TestRegistry::registerTest( TestCase const& testCase ) {
14303     std::string name = testCase.getTestCaseInfo().name;
14304     if( name.empty() ) {
14305         ReusableStringStream rss;
14306         rss << "Anonymous test case " << ++m_unnamedCount;
14307         return registerTest( testCase.withName( rss.str() ) );
14308     }
14309     m_functions.push_back( testCase );
14310 }
14311
14312 std::vector<TestCase> const& TestRegistry::getAllTests() const {
14313     return m_functions;
14314 }
14315 std::vector<TestCase> const& TestRegistry::getAllTestsSorted( IConfig const& config ) const {
14316     if( m_sortedFunctions.empty() )
14317         enforceNoDuplicateTestCases( m_functions );
14318
14319     if( m_currentSortOrder != config.runOrder() || m_sortedFunctions.empty() ) {
14320         m_sortedFunctions = sortTests( config, m_functions );
14321     }

```

```

14321         m_currentSortOrder = config.runOrder();
14322     }
14323     return m_sortedFunctions;
14324 }
14325
14327     TestInvokerAsFunction::TestInvokerAsFunction( void(*testAsFunction)() ) noexcept :
m_testAsFunction( testAsFunction ) {}
14328
14329     void TestInvokerAsFunction::invoke() const {
14330         m_testAsFunction();
14331     }
14332
14333     std::string extractClassName( StringRef const& classOrQualifiedMethodName ) {
14334         std::string className(classOrQualifiedMethodName);
14335         if( startsWith( className, '&' ) )
14336         {
14337             std::size_t lastColons = className.rfind( "::" );
14338             std::size_t penultimateColons = className.rfind( "::", lastColons-1 );
14339             if( penultimateColons == std::string::npos )
14340                 penultimateColons = 1;
14341             className = className.substr( penultimateColons, lastColons-penultimateColons );
14342         }
14343         return className;
14344     }
14345
14346 } // end namespace Catch
14347 // end catch_test_case_registry_impl.cpp
14348 // start catch_test_case_tracker.cpp
14349
14350 #include <algorithm>
14351 #include <cassert>
14352 #include <stdexcept>
14353 #include <memory>
14354 #include <sstream>
14355
14356 #if defined(__clang__)
14357 #   pragma clang diagnostic push
14358 #   pragma clang diagnostic ignored "-Wexit-time-destructors"
14359 #endif
14360
14361 namespace Catch {
14362     namespace TestCaseTracking {
14363
14364         NameAndLocation::NameAndLocation( std::string const& _name, SourceLineInfo const& _location )
14365         :   name( _name ),
14366             location( _location )
14367         {}
14368
14369         ITracker::~ITracker() = default;
14370
14371         ITracker& TrackerContext::startRun() {
14372             m_rootTracker = std::make_shared<SectionTracker>( NameAndLocation( "{root}",
CATCH_INTERNAL_LINEINFO ), *this, nullptr );
14373             m_currentTracker = nullptr;
14374             m_runState = Executing;
14375             return *m_rootTracker;
14376         }
14377
14378         void TrackerContext::endRun() {
14379             m_rootTracker.reset();
14380             m_currentTracker = nullptr;
14381             m_runState = NotStarted;
14382         }
14383
14384         void TrackerContext::startCycle() {
14385             m_currentTracker = m_rootTracker.get();
14386             m_runState = Executing;
14387         }
14388         void TrackerContext::completeCycle() {
14389             m_runState = CompletedCycle;
14390         }
14391
14392         bool TrackerContext::completedCycle() const {
14393             return m_runState == CompletedCycle;
14394         }
14395         ITracker& TrackerContext::currentTracker() {
14396             return *m_currentTracker;
14397         }
14398         void TrackerContext::setCurrentTracker( ITracker* tracker ) {
14399             m_currentTracker = tracker;
14400         }
14401
14402         TrackerBase::TrackerBase( NameAndLocation const& nameAndLocation, TrackerContext& ctx, ITracker*
parent ) :
14403             ITracker( nameAndLocation ),
14404             m_ctx( ctx ),
14405             m_parent( parent )

```

```

14406     {}
14407
14408     bool TrackerBase::isComplete() const {
14409         return m_runState == CompletedSuccessfully || m_runState == Failed;
14410     }
14411     bool TrackerBase::isSuccessfullyCompleted() const {
14412         return m_runState == CompletedSuccessfully;
14413     }
14414     bool TrackerBase::isOpen() const {
14415         return m_runState != NotStarted && !isComplete();
14416     }
14417     bool TrackerBase::hasChildren() const {
14418         return !m_children.empty();
14419     }
14420
14421     void TrackerBase::addChild( ITrackerPtr const& child ) {
14422         m_children.push_back( child );
14423     }
14424
14425     ITrackerPtr TrackerBase::findChild( NameAndLocation const& nameAndLocation ) {
14426         auto it = std::find_if( m_children.begin(), m_children.end(),
14427             [&nameAndLocation]( ITrackerPtr const& tracker ) {
14428                 return
14429                     tracker->nameAndLocation().location == nameAndLocation.location &&
14430                     tracker->nameAndLocation().name == nameAndLocation.name;
14431             } );
14432         return( it != m_children.end() )
14433             ? *it
14434             : nullptr;
14435     }
14436     ITracker& TrackerBase::parent() {
14437         assert( m_parent ); // Should always be non-null except for root
14438         return *m_parent;
14439     }
14440
14441     void TrackerBase::openChild() {
14442         if( m_runState != ExecutingChildren ) {
14443             m_runState = ExecutingChildren;
14444             if( m_parent )
14445                 m_parent->openChild();
14446         }
14447     }
14448
14449     bool TrackerBase::isSectionTracker() const { return false; }
14450     bool TrackerBase::isGeneratorTracker() const { return false; }
14451
14452     void TrackerBase::open() {
14453         m_runState = Executing;
14454         moveToThis();
14455         if( m_parent )
14456             m_parent->openChild();
14457     }
14458
14459     void TrackerBase::close() {
14460
14461         // Close any still open children (e.g. generators)
14462         while( &m_ctx.currentTracker() != this )
14463             m_ctx.currentTracker().close();
14464
14465         switch( m_runState ) {
14466             case NeedsAnotherRun:
14467                 break;
14468
14469             case Executing:
14470                 m_runState = CompletedSuccessfully;
14471                 break;
14472             case ExecutingChildren:
14473                 if( std::all_of(m_children.begin(), m_children.end(), [](ITrackerPtr const& t){ return
14474 t->isComplete(); }) )
14475                     m_runState = CompletedSuccessfully;
14476                 break;
14477             case NotStarted:
14478             case CompletedSuccessfully:
14479             case Failed:
14480                 CATCH_INTERNAL_ERROR( "Illogical state: " << m_runState );
14481
14482             default:
14483                 CATCH_INTERNAL_ERROR( "Unknown state: " << m_runState );
14484         }
14485         moveToParent();
14486         m_ctx.completeCycle();
14487     }
14488     void TrackerBase::fail() {
14489         m_runState = Failed;
14490         if( m_parent )
14491             m_parent->markAsNeedingAnotherRun();

```

```

14492         moveToParent();
14493         m_ctx.completeCycle();
14494     }
14495     void TrackerBase::markAsNeedingAnotherRun() {
14496         m_runState = NeedsAnotherRun;
14497     }
14498
14499     void TrackerBase::moveToParent() {
14500         assert( m_parent );
14501         m_ctx.setCurrentTracker( m_parent );
14502     }
14503     void TrackerBase::moveToThis() {
14504         m_ctx.setCurrentTracker( this );
14505     }
14506
14507     SectionTracker::SectionTracker( NameAndLocation const& nameAndLocation, TrackerContext& ctx,
ITracker* parent )
14508     :   TrackerBase( nameAndLocation, ctx, parent ),
14509         m_trimmed_name(trim(nameAndLocation.name))
14510     {
14511         if( parent ) {
14512             while( !parent->isSectionTracker() )
14513                 parent = &parent->parent();
14514
14515             SectionTracker& parentSection = static_cast<SectionTracker&>( *parent );
14516             addNextFilters( parentSection.m_filters );
14517         }
14518     }
14519
14520     bool SectionTracker::isComplete() const {
14521         bool complete = true;
14522
14523         if (m_filters.empty())
14524             || m_filters[0] == ""
14525             || std::find(m_filters.begin(), m_filters.end(), m_trimmed_name) != m_filters.end()) {
14526             complete = TrackerBase::isComplete();
14527         }
14528         return complete;
14529     }
14530
14531     bool SectionTracker::isSectionTracker() const { return true; }
14532
14533     SectionTracker& SectionTracker::acquire( TrackerContext& ctx, NameAndLocation const&
nameAndLocation ) {
14534         std::shared_ptr<SectionTracker> section;
14535
14536         ITracker& currentTracker = ctx.currentTracker();
14537         if( ITrackerPtr childTracker = currentTracker.findChild( nameAndLocation ) ) {
14538             assert( childTracker );
14539             assert( childTracker->isSectionTracker() );
14540             section = std::static_pointer_cast<SectionTracker>( childTracker );
14541         }
14542         else {
14543             section = std::make_shared<SectionTracker>( nameAndLocation, ctx, &currentTracker );
14544             currentTracker.addChild( section );
14545         }
14546         if( !ctx.completedCycle() )
14547             section->tryOpen();
14548         return *section;
14549     }
14550
14551     void SectionTracker::tryOpen() {
14552         if( !isComplete() )
14553             open();
14554     }
14555
14556     void SectionTracker::addInitialFilters( std::vector<std::string> const& filters ) {
14557         if( !filters.empty() ) {
14558             m_filters.reserve( m_filters.size() + filters.size() + 2 );
14559             m_filters.emplace_back(""); // Root - should never be consulted
14560             m_filters.emplace_back(""); // Test Case - not a section filter
14561             m_filters.insert( m_filters.end(), filters.begin(), filters.end() );
14562         }
14563     }
14564     void SectionTracker::addNextFilters( std::vector<std::string> const& filters ) {
14565         if( filters.size() > 1 )
14566             m_filters.insert( m_filters.end(), filters.begin()+1, filters.end() );
14567     }
14568
14569     std::vector<std::string> const& SectionTracker::getFilters() const {
14570         return m_filters;
14571     }
14572
14573     std::string const& SectionTracker::trimmedName() const {
14574         return m_trimmed_name;
14575     }
14576

```

```

14577 } // namespace TestCaseTracking
14578
14579 using TestCaseTracking::ITracker;
14580 using TestCaseTracking::TrackerContext;
14581 using TestCaseTracking::SectionTracker;
14582
14583 } // namespace Catch
14584
14585 #if defined(__clang__)
14586 # pragma clang diagnostic pop
14587 #endif
14588 // end catch_test_case_tracker.cpp
14589 // start catch_test_registry.cpp
14590
14591 namespace Catch {
14592
14593     auto makeTestInvoker( void(*testAsFunction)() ) noexcept -> ITestInvoker* {
14594         return new(std::nothrow) TestInvokerAsFunction( testAsFunction );
14595     }
14596
14597     NameAndTags::NameAndTags( StringRef const& name_ , StringRef const& tags_ ) noexcept : name( name_
), tags( tags_ ) {}
14598
14599     AutoReg::AutoReg( ITestInvoker* invoker, SourceLineInfo const& lineInfo, StringRef const&
classOrMethod, NameAndTags const& nameAndTags ) noexcept {
14600         CATCH_TRY {
14601             getMutableRegistryHub()
14602                 .registerTest(
14603                     makeTestCase(
14604                         invoker,
14605                         extractClassName( classOrMethod ),
14606                         nameAndTags,
14607                         lineInfo));
14608         } CATCH_CATCH_ALL {
14609             // Do not throw when constructing global objects, instead register the exception to be
            processed later
14610             getMutableRegistryHub().registerStartupException();
14611         }
14612     }
14613
14614     AutoReg::~AutoReg() = default;
14615 }
14616 // end catch_test_registry.cpp
14617 // start catch_test_spec.cpp
14618
14619 #include <algorithm>
14620 #include <string>
14621 #include <vector>
14622 #include <memory>
14623
14624 namespace Catch {
14625
14626     TestSpec::Pattern::Pattern( std::string const& name )
14627     : m_name( name )
14628     {}
14629
14630     TestSpec::Pattern::~Pattern() = default;
14631
14632     std::string const& TestSpec::Pattern::name() const {
14633         return m_name;
14634     }
14635
14636     TestSpec::NamePattern::NamePattern( std::string const& name, std::string const& filterString )
14637     : Pattern( filterString )
14638     , m_wildcardPattern( toLower( name ), CaseSensitive::No )
14639     {}
14640
14641     bool TestSpec::NamePattern::matches( TestCaseInfo const& testCase ) const {
14642         return m_wildcardPattern.matches( testCase.name );
14643     }
14644
14645     TestSpec::TagPattern::TagPattern( std::string const& tag, std::string const& filterString )
14646     : Pattern( filterString )
14647     , m_tag( toLower( tag ) )
14648     {}
14649
14650     bool TestSpec::TagPattern::matches( TestCaseInfo const& testCase ) const {
14651         return std::find( begin( testCase.lcaseTags ),
14652                         end( testCase.lcaseTags ),
14653                         m_tag ) != end( testCase.lcaseTags );
14654     }
14655
14656     TestSpec::ExcludedPattern::ExcludedPattern( PatternPtr const& underlyingPattern )
14657     : Pattern( underlyingPattern->name() )
14658     , m_underlyingPattern( underlyingPattern )
14659     {}
14660

```

```

14661     bool TestSpec::ExcludedPattern::matches( TestCaseInfo const& testCase ) const {
14662         return !m_underlyingPattern->matches( testCase );
14663     }
14664
14665     bool TestSpec::Filter::matches( TestCaseInfo const& testCase ) const {
14666         return std::all_of( m_patterns.begin(), m_patterns.end(), [&]( PatternPtr const& p ){ return
p->matches( testCase ); } );
14667     }
14668
14669     std::string TestSpec::Filter::name() const {
14670         std::string name;
14671         for( auto const& p : m_patterns )
14672             name += p->name();
14673         return name;
14674     }
14675
14676     bool TestSpec::hasFilters() const {
14677         return !m_filters.empty();
14678     }
14679
14680     bool TestSpec::matches( TestCaseInfo const& testCase ) const {
14681         return std::any_of( m_filters.begin(), m_filters.end(), [&]( Filter const& f ){ return
f.matches( testCase ); } );
14682     }
14683
14684     TestSpec::Matches TestSpec::matchesByFilter( std::vector<TestCase> const& testCases, IConfig
const& config ) const
14685     {
14686         Matches matches( m_filters.size() );
14687         std::transform( m_filters.begin(), m_filters.end(), matches.begin(), [&]( Filter const& filter
){
14688             std::vector<TestCase const*> currentMatches;
14689             for( auto const& test : testCases )
14690                 if( isThrowSafe( test, config ) && filter.matches( test ) )
14691                     currentMatches.emplace_back( &test );
14692             return FilterMatch{ filter.name(), currentMatches };
14693         } );
14694         return matches;
14695     }
14696
14697     const TestSpec::vectorStrings& TestSpec::getInvalidArgs() const{
14698         return (m_invalidArgs);
14699     }
14700 }
14701 // end catch_test_spec.cpp
14702 // start catch_test_spec_parser.cpp
14703
14704 namespace Catch {
14705
14706     TestSpecParser::TestSpecParser( ITagAliasRegistry const& tagAliases ) : m_tagAliases( &tagAliases
) {}
14707
14708     TestSpecParser& TestSpecParser::parse( std::string const& arg ) {
14709         m_mode = None;
14710         m_exclusion = false;
14711         m_arg = m_tagAliases->expandAliases( arg );
14712         m_escapeChars.clear();
14713         m_substring.reserve(m_arg.size());
14714         m_patternName.reserve(m_arg.size());
14715         m_realPatternPos = 0;
14716
14717         for( m_pos = 0; m_pos < m_arg.size(); ++m_pos )
14718             //if visitChar fails
14719             if( !visitChar( m_arg[m_pos] ) ){
14720                 if( !visitChar( m_arg[m_pos] ) ){
14721                     m_testSpec.m_invalidArgs.push_back( arg );
14722                     break;
14723                 }
14724             }
14725         return *this;
14726     }
14727
14728     TestSpec TestSpecParser::testSpec() {
14729         addFilter();
14730         return m_testSpec;
14731     }
14732
14733     bool TestSpecParser::visitChar( char c ) {
14734         if( (m_mode != EscapedName) && (c == '\\') ) {
14735             escape();
14736             addCharToPattern(c);
14737             return true;
14738         } else if( (m_mode != EscapedName) && (c == ',') ) {
14739             return separate();
14740         }
14741
14742         switch( m_mode ) {
14743             case None:
14744                 if( processNoneChar( c ) )

```

```

14743         return true;
14744     break;
14745     case Name:
14746         processNameChar( c );
14747         break;
14748     case EscapedName:
14749         endMode();
14750         addCharToPattern(c);
14751         return true;
14752     default:
14753     case Tag:
14754     case QuotedName:
14755         if( processOtherChar( c ) )
14756             return true;
14757         break;
14758     }
14759
14760     m_substring += c;
14761     if( !isControlChar( c ) ) {
14762         m_patternName += c;
14763         m_realPatternPos++;
14764     }
14765     return true;
14766 }
14767 // Two of the processing methods return true to signal the caller to return
14768 // without adding the given character to the current pattern strings
14769 bool TestSpecParser::processNoneChar( char c ) {
14770     switch( c ) {
14771     case ' ':
14772         return true;
14773     case '~':
14774         m_exclusion = true;
14775         return false;
14776     case '[':
14777         startNewMode( Tag );
14778         return false;
14779     case '"':
14780         startNewMode( QuotedName );
14781         return false;
14782     default:
14783         startNewMode( Name );
14784         return false;
14785     }
14786 }
14787 void TestSpecParser::processNameChar( char c ) {
14788     if( c == '[' ) {
14789         if( m_substring == "exclude:" )
14790             m_exclusion = true;
14791         else
14792             endMode();
14793         startNewMode( Tag );
14794     }
14795 }
14796 bool TestSpecParser::processOtherChar( char c ) {
14797     if( !isControlChar( c ) )
14798         return false;
14799     m_substring += c;
14800     endMode();
14801     return true;
14802 }
14803 void TestSpecParser::startNewMode( Mode mode ) {
14804     m_mode = mode;
14805 }
14806 void TestSpecParser::endMode() {
14807     switch( m_mode ) {
14808     case Name:
14809     case QuotedName:
14810         return addNamePattern();
14811     case Tag:
14812         return addTagPattern();
14813     case EscapedName:
14814         revertBackToLastMode();
14815         return;
14816     case None:
14817     default:
14818         return startNewMode( None );
14819     }
14820 }
14821 void TestSpecParser::escape() {
14822     saveLastMode();
14823     m_mode = EscapedName;
14824     m_escapeChars.push_back( m_realPatternPos );
14825 }
14826 bool TestSpecParser::isControlChar( char c ) const {
14827     switch( m_mode ) {
14828     default:
14829         return false;

```

```

14830         case None:
14831             return c == '~';
14832         case Name:
14833             return c == '[';
14834         case EscapedName:
14835             return true;
14836         case QuotedName:
14837             return c == '"';
14838         case Tag:
14839             return c == '[' || c == ']';
14840     }
14841 }
14842
14843 void TestSpecParser::addFilter() {
14844     if( !m_currentFilter.m_patterns.empty() ) {
14845         m_testSpec.m_filters.push_back( m_currentFilter );
14846         m_currentFilter = TestSpec::Filter();
14847     }
14848 }
14849
14850 void TestSpecParser::saveLastMode() {
14851     lastMode = m_mode;
14852 }
14853
14854 void TestSpecParser::revertBackToLastMode() {
14855     m_mode = lastMode;
14856 }
14857
14858 bool TestSpecParser::separate() {
14859     if( (m_mode==QuotedName) || (m_mode==Tag) ){
14860         //invalid argument, signal failure to previous scope.
14861         m_mode = None;
14862         m_pos = m_arg.size();
14863         m_substring.clear();
14864         m_patternName.clear();
14865         m_realPatternPos = 0;
14866         return false;
14867     }
14868     endMode();
14869     addFilter();
14870     return true; //success
14871 }
14872
14873 std::string TestSpecParser::preprocessPattern() {
14874     std::string token = m_patternName;
14875     for (std::size_t i = 0; i < m_escapeChars.size(); ++i)
14876         token = token.substr(0, m_escapeChars[i] - i) + token.substr(m_escapeChars[i] - i + 1);
14877     m_escapeChars.clear();
14878     if (startsWith(token, "exclude:")) {
14879         m_exclusion = true;
14880         token = token.substr(8);
14881     }
14882
14883     m_patternName.clear();
14884     m_realPatternPos = 0;
14885
14886     return token;
14887 }
14888
14889 void TestSpecParser::addNamePattern() {
14890     auto token = preprocessPattern();
14891
14892     if (!token.empty()) {
14893         TestSpec::PatternPtr pattern = std::make_shared<TestSpec::NamePattern>(token,
14894 m_substring);
14895         if (m_exclusion)
14896             pattern = std::make_shared<TestSpec::ExcludedPattern>(pattern);
14897         m_currentFilter.m_patterns.push_back(pattern);
14898     }
14899     m_substring.clear();
14900     m_exclusion = false;
14901     m_mode = None;
14902 }
14903
14904 void TestSpecParser::addTagPattern() {
14905     auto token = preprocessPattern();
14906
14907     if (!token.empty()) {
14908         // If the tag pattern is the "hide and tag" shorthand (e.g. [.foo])
14909         // we have to create a separate hide tag and shorten the real one
14910         if (token.size() > 1 && token[0] == '.') {
14911             token.erase(token.begin());
14912             TestSpec::PatternPtr pattern = std::make_shared<TestSpec::TagPattern>(".",
14913 m_substring);
14914             if (m_exclusion) {
14915                 pattern = std::make_shared<TestSpec::ExcludedPattern>(pattern);
14916             }
14917         }
14918     }

```



```

14915         m_currentFilter.m_patterns.push_back(pattern);
14916     }
14917
14918     TestSpec::PatternPtr pattern = std::make_shared<TestSpec::TagPattern>(token, m_substring);
14919
14920     if (m_exclusion) {
14921         pattern = std::make_shared<TestSpec::ExcludedPattern>(pattern);
14922     }
14923     m_currentFilter.m_patterns.push_back(pattern);
14924 }
14925 m_substring.clear();
14926 m_exclusion = false;
14927 m_mode = None;
14928 }
14929
14930 TestSpec parseTestSpec( std::string const& arg ) {
14931     return TestSpecParser( ITagAliasRegistry::get() ).parse( arg ).testSpec();
14932 }
14933
14934 } // namespace Catch
14935 // end catch_test_spec_parser.cpp
14936 // start catch_timer.cpp
14937
14938 #include <chrono>
14939
14940 static const uint64_t nanosecondsInSecond = 1000000000;
14941
14942 namespace Catch {
14943
14944     auto getCurrentNanosecondsSinceEpoch() -> uint64_t {
14945         return std::chrono::duration_cast<std::chrono::nanoseconds>(
14946             std::chrono::high_resolution_clock::now().time_since_epoch() ).count();
14947     }
14948
14949     namespace {
14950         auto estimateClockResolution() -> uint64_t {
14951             uint64_t sum = 0;
14952             static const uint64_t iterations = 1000000;
14953
14954             auto startTime = getCurrentNanosecondsSinceEpoch();
14955
14956             for( std::size_t i = 0; i < iterations; ++i ) {
14957                 uint64_t ticks;
14958                 uint64_t baseTicks = getCurrentNanosecondsSinceEpoch();
14959                 do {
14960                     ticks = getCurrentNanosecondsSinceEpoch();
14961                 } while( ticks == baseTicks );
14962
14963                 auto delta = ticks - baseTicks;
14964                 sum += delta;
14965
14966                 // If we have been calibrating for over 3 seconds -- the clock
14967                 // is terrible and we should move on.
14968                 // TBD: How to signal that the measured resolution is probably wrong?
14969                 if (ticks > startTime + 3 * nanosecondsInSecond) {
14970                     return sum / ( i + 1u );
14971                 }
14972             }
14973
14974             // We're just taking the mean, here. To do better we could take the std. dev and exclude
14975             // outliers
14976             // - and potentially do more iterations if there's a high variance.
14977             return sum/iterations;
14978         }
14979
14980         auto getEstimatedClockResolution() -> uint64_t {
14981             static auto s_resolution = estimateClockResolution();
14982             return s_resolution;
14983         }
14984
14985         void Timer::start() {
14986             m_nanoseconds = getCurrentNanosecondsSinceEpoch();
14987         }
14988         auto Timer::getElapsedNanoseconds() const -> uint64_t {
14989             return getCurrentNanosecondsSinceEpoch() - m_nanoseconds;
14990         }
14991         auto Timer::getElapsedMicroseconds() const -> uint64_t {
14992             return getElapsedNanoseconds()/1000;
14993         }
14994         auto Timer::getElapsedMilliseconds() const -> unsigned int {
14995             return static_cast<unsigned int>(getElapsedMicroseconds()/1000);
14996         }
14997         auto Timer::getElapsedSeconds() const -> double {
14998             return getElapsedMicroseconds()/1000000.0;
14999         }
15000     }

```

```

15000 } // namespace Catch
15001 // end catch_timer.cpp
15002 // start catch_tostring.cpp
15003
15004 #if defined(__clang__)
15005 #   pragma clang diagnostic push
15006 #   pragma clang diagnostic ignored "-Wexit-time-destructors"
15007 #   pragma clang diagnostic ignored "-Wglobal-constructors"
15008 #endif
15009
15010 // Enable specific decls locally
15011 #if !defined(CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER)
15012 #define CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER
15013 #endif
15014
15015 #include <cmath>
15016 #include <iomanip>
15017
15018 namespace Catch {
15019     namespace Detail {
15020
15021         const std::string unprintableString = "{?}";
15022
15023         namespace {
15024             const int hexThreshold = 255;
15025
15026             struct Endianness {
15027                 enum Arch { Big, Little };
15028
15029                 static Arch which() {
15030                     int one = 1;
15031                     // If the lowest byte we read is non-zero, we can assume
15032                     // that little endian format is used.
15033                     auto value = *reinterpret_cast<char*>(&one);
15034                     return value ? Little : Big;
15035                 }
15036             };
15037         }
15038
15039         std::string rawMemoryToString( const void *object, std::size_t size ) {
15040             // Reverse order for little endian architectures
15041             int i = 0, end = static_cast<int>( size ), inc = 1;
15042             if( Endianness::which() == Endianness::Little ) {
15043                 i = end-1;
15044                 end = inc = -1;
15045             }
15046
15047             unsigned char const *bytes = static_cast<unsigned char const *>(object);
15048             ReusableStringStream rss;
15049             rss << "0x" << std::setfill('0') << std::hex;
15050             for( ; i != end; i += inc )
15051                 rss << std::setw(2) << static_cast<unsigned>(bytes[i]);
15052             return rss.str();
15053         }
15054     }
15055 }
15056
15057 template<typename T>
15058 std::string fpToString( T value, int precision ) {
15059     if (Catch::isnan(value)) {
15060         return "nan";
15061     }
15062
15063     ReusableStringStream rss;
15064     rss << std::setprecision( precision )
15065         << std::fixed
15066         << value;
15067     std::string d = rss.str();
15068     std::size_t i = d.find_last_not_of( '0' );
15069     if( i != std::string::npos && i != d.size()-1 ) {
15070         if( d[i] == '.' )
15071             i++;
15072         d = d.substr( 0, i+1 );
15073     }
15074     return d;
15075 }
15076
15077 //
15078 // Out-of-line defs for full specialization of StringMaker
15079 //
15080
15081 std::string StringMaker<std::string>::convert(const std::string& str) {
15082     if (!getCurrentContext().getConfig()->showInvisibles()) {
15083         return "'" + str + "'";
15084     }
15085     std::string s("\");

```

```

15089     for (char c : str) {
15090         switch (c) {
15091             case '\n':
15092                 s.append("\n");
15093                 break;
15094             case '\t':
15095                 s.append("\t");
15096                 break;
15097             default:
15098                 s.push_back(c);
15099                 break;
15100         }
15101     }
15102     s.append("\"");
15103     return s;
15104 }
15105
15106 #ifndef CATCH_CONFIG_CPP17_STRING_VIEW
15107 std::string StringMaker<std::string_view>::convert(std::string_view str) {
15108     return ::Catch::Detail::stringify(std::string{ str });
15109 }
15110 #endif
15111
15112 std::string StringMaker<char const*>::convert(char const* str) {
15113     if (str) {
15114         return ::Catch::Detail::stringify(std::string{ str });
15115     } else {
15116         return "{null string}";
15117     }
15118 }
15119
15120 std::string StringMaker<char*>::convert(char* str) {
15121     if (str) {
15122         return ::Catch::Detail::stringify(std::string{ str });
15123     } else {
15124         return "{null string}";
15125     }
15126 }
15127
15128 #ifndef CATCH_CONFIG_WCHAR
15129 std::string StringMaker<std::wstring>::convert(const std::wstring& wstr) {
15130     std::string s;
15131     s.reserve(wstr.size());
15132     for (auto c : wstr) {
15133         s += (c <= 0xff) ? static_cast<char>(c) : '?';
15134     }
15135     return ::Catch::Detail::stringify(s);
15136 }
15137 #endif
15138
15139 #ifdef CATCH_CONFIG_CPP17_STRING_VIEW
15140 std::string StringMaker<std::wstring_view>::convert(std::wstring_view str) {
15141     return StringMaker<std::wstring>::convert(std::wstring(str));
15142 }
15143 #endif
15144
15145 std::string StringMaker<wchar_t const*>::convert(wchar_t const* str) {
15146     if (str) {
15147         return ::Catch::Detail::stringify(std::wstring{ str });
15148     } else {
15149         return "{null string}";
15150     }
15151 }
15152
15153 std::string StringMaker<wchar_t*>::convert(wchar_t* str) {
15154     if (str) {
15155         return ::Catch::Detail::stringify(std::wstring{ str });
15156     } else {
15157         return "{null string}";
15158     }
15159 }
15160 #endif
15161
15162 #if defined(CATCH_CONFIG_CPP17_BYTE)
15163 #include <cstdint>
15164 std::string StringMaker<std::byte>::convert(std::byte value) {
15165     return ::Catch::Detail::stringify(std::to_integer<unsigned long long>(value));
15166 }
15167 #endif // defined(CATCH_CONFIG_CPP17_BYTE)
15168
15169 std::string StringMaker<int>::convert(int value) {
15170     return ::Catch::Detail::stringify(static_cast<long long>(value));
15171 }
15172
15173 std::string StringMaker<long>::convert(long value) {
15174     return ::Catch::Detail::stringify(static_cast<long long>(value));
15175 }
15176
15177 std::string StringMaker<long long>::convert(long long value) {
15178     ReusableStringStream rss;
15179     rss << value;
15180     if (value > Detail::hexThreshold) {

```

```

15176         rss << " (0x" << std::hex << value << ')';
15177     }
15178     return rss.str();
15179 }
15180
15181 std::string StringMaker<unsigned int>::convert(unsigned int value) {
15182     return ::Catch::Detail::stringify(static_cast<unsigned long long>(value));
15183 }
15184 std::string StringMaker<unsigned long>::convert(unsigned long value) {
15185     return ::Catch::Detail::stringify(static_cast<unsigned long long>(value));
15186 }
15187 std::string StringMaker<unsigned long long>::convert(unsigned long long value) {
15188     ReusableStringStream rss;
15189     rss << value;
15190     if (value > Detail::hexThreshold) {
15191         rss << " (0x" << std::hex << value << ')';
15192     }
15193     return rss.str();
15194 }
15195
15196 std::string StringMaker<bool>::convert(bool b) {
15197     return b ? "true" : "false";
15198 }
15199
15200 std::string StringMaker<signed char>::convert(signed char value) {
15201     if (value == '\r') {
15202         return "\\r";
15203     } else if (value == '\f') {
15204         return "\\f";
15205     } else if (value == '\n') {
15206         return "\\n";
15207     } else if (value == '\t') {
15208         return "\\t";
15209     } else if ('\0' <= value && value < ' ') {
15210         return ::Catch::Detail::stringify(static_cast<unsigned int>(value));
15211     } else {
15212         char chstr[] = " ";
15213         chstr[1] = value;
15214         return chstr;
15215     }
15216 }
15217 std::string StringMaker<char>::convert(char c) {
15218     return ::Catch::Detail::stringify(static_cast<signed char>(c));
15219 }
15220 std::string StringMaker<unsigned char>::convert(unsigned char c) {
15221     return ::Catch::Detail::stringify(static_cast<char>(c));
15222 }
15223
15224 std::string StringMaker<std::nullptr_t>::convert(std::nullptr_t) {
15225     return "nullptr";
15226 }
15227
15228 int StringMaker<float>::precision = 5;
15229
15230 std::string StringMaker<float>::convert(float value) {
15231     return fpToString(value, precision) + 'f';
15232 }
15233
15234 int StringMaker<double>::precision = 10;
15235
15236 std::string StringMaker<double>::convert(double value) {
15237     return fpToString(value, precision);
15238 }
15239
15240 std::string ratio_string<std::atto>::symbol() { return "a"; }
15241 std::string ratio_string<std::femto>::symbol() { return "f"; }
15242 std::string ratio_string<std::pico>::symbol() { return "p"; }
15243 std::string ratio_string<std::nano>::symbol() { return "n"; }
15244 std::string ratio_string<std::micro>::symbol() { return "u"; }
15245 std::string ratio_string<std::milli>::symbol() { return "m"; }
15246
15247 } // end namespace Catch
15248
15249 #if defined(__clang__)
15250 #    pragma clang diagnostic pop
15251 #endif
15252
15253 // end catch_tostring.cpp
15254 // start catch_totals.cpp
15255
15256 namespace Catch {
15257
15258     Counts Counts::operator - ( Counts const& other ) const {
15259         Counts diff;
15260         diff.passed = passed - other.passed;
15261         diff.failed = failed - other.failed;
15262         diff.failedButOk = failedButOk - other.failedButOk;

```

```

15263         return diff;
15264     }
15265
15266     Counts& Counts::operator += ( Counts const& other ) {
15267         passed += other.passed;
15268         failed += other.failed;
15269         failedButOk += other.failedButOk;
15270         return *this;
15271     }
15272
15273     std::size_t Counts::total() const {
15274         return passed + failed + failedButOk;
15275     }
15276     bool Counts::allPassed() const {
15277         return failed == 0 && failedButOk == 0;
15278     }
15279     bool Counts::allOk() const {
15280         return failed == 0;
15281     }
15282
15283     Totals Totals::operator - ( Totals const& other ) const {
15284         Totals diff;
15285         diff.assertions = assertions - other.assertions;
15286         diff.testCases = testCases - other.testCases;
15287         return diff;
15288     }
15289
15290     Totals& Totals::operator += ( Totals const& other ) {
15291         assertions += other.assertions;
15292         testCases += other.testCases;
15293         return *this;
15294     }
15295
15296     Totals Totals::delta( Totals const& prevTotals ) const {
15297         Totals diff = *this - prevTotals;
15298         if( diff.assertions.failed > 0 )
15299             ++diff.testCases.failed;
15300         else if( diff.assertions.failedButOk > 0 )
15301             ++diff.testCases.failedButOk;
15302         else
15303             ++diff.testCases.passed;
15304         return diff;
15305     }
15306 }
15307
15308 // end catch_totals.cpp
15309 // start catch_uncaught_exceptions.cpp
15310
15311 // start catch_config_uncaught_exceptions.hpp
15312
15313 // Copyright Catch2 Authors
15314 // Distributed under the Boost Software License, Version 1.0.
15315 // (See accompanying file LICENSE_1_0.txt or copy at
15316 //  https://www.boost.org/LICENSE\_1\_0.txt)
15317
15318 // SPDX-License-Identifier: BSL-1.0
15319
15320 #ifndef CATCH_CONFIG_UNCAUGHT_EXCEPTIONS_HPP
15321 #define CATCH_CONFIG_UNCAUGHT_EXCEPTIONS_HPP
15322
15323 #if defined(_MSC_VER)
15324 #   if _MSC_VER >= 1900 // Visual Studio 2015 or newer
15325 #       define CATCH_INTERNAL_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS
15326 #   endif
15327 #endif
15328
15329 #include <exception>
15330
15331 #if defined(__cpp_lib_uncaught_exceptions) \
15332     && !defined(CATCH_INTERNAL_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS)
15333
15334 #   define CATCH_INTERNAL_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS
15335 #endif // __cpp_lib_uncaught_exceptions
15336
15337 #if defined(CATCH_INTERNAL_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS) \
15338     && !defined(CATCH_CONFIG_NO_CPP17_UNCAUGHT_EXCEPTIONS) \
15339     && !defined(CATCH_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS)
15340
15341 #   define CATCH_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS
15342 #endif
15343
15344 #endif // CATCH_CONFIG_UNCAUGHT_EXCEPTIONS_HPP
15345 // end catch_config_uncaught_exceptions.hpp
15346 #include <exception>
15347
15348 namespace Catch {
15349     bool uncaught_exceptions() {

```

```

15350 #if defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
15351     return false;
15352 #elif defined(CATCH_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS)
15353     return std::uncaught_exceptions() > 0;
15354 #else
15355     return std::uncaught_exception();
15356 #endif
15357 }
15358 } // end namespace Catch
15359 // end catch_uncaught_exceptions.cpp
15360 // start catch_version.cpp
15361
15362 #include <ostream>
15363
15364 namespace Catch {
15365     Version::Version
15366     (   unsigned int _majorVersion,
15367         unsigned int _minorVersion,
15368         unsigned int _patchNumber,
15369         char const * _branchName,
15370         unsigned int _buildNumber )
15371     :   majorVersion( _majorVersion ),
15372         minorVersion( _minorVersion ),
15373         patchNumber( _patchNumber ),
15374         branchName( _branchName ),
15375         buildNumber( _buildNumber )
15376     {}
15377
15378     std::ostream& operator << ( std::ostream& os, Version const& version ) {
15379         os << version.majorVersion << '.'
15380            << version.minorVersion << '.'
15381            << version.patchNumber;
15382         // branchName is never null -> 0th char is \0 if it is empty
15383         if (version.branchName[0]) {
15384             os << '-' << version.branchName
15385                << '.' << version.buildNumber;
15386         }
15387         return os;
15388     }
15389 }
15390
15391 Version const& libraryVersion() {
15392     static Version version( 2, 13, 10, "", 0 );
15393     return version;
15394 }
15395
15396 }
15397 // end catch_version.cpp
15398 // start catch_wildcard_pattern.cpp
15399
15400 namespace Catch {
15401     WildcardPattern::WildcardPattern( std::string const& pattern,
15402                                       CaseSensitive::Choice caseSensitivity )
15403     :   m_caseSensitivity( caseSensitivity ),
15404         m_pattern( normaliseString( pattern ) )
15405     {
15406         if ( startsWith( m_pattern, '*' ) ) {
15407             m_pattern = m_pattern.substr( 1 );
15408             m_wildcard = WildcardAtStart;
15409         }
15410         if ( endsWith( m_pattern, '*' ) ) {
15411             m_pattern = m_pattern.substr( 0, m_pattern.size()-1 );
15412             m_wildcard = static_cast<WildcardPosition>( m_wildcard | WildcardAtEnd );
15413         }
15414     }
15415
15416     bool WildcardPattern::matches( std::string const& str ) const {
15417         switch( m_wildcard ) {
15418             case NoWildcard:
15419                 return m_pattern == normaliseString( str );
15420             case WildcardAtStart:
15421                 return endsWith( normaliseString( str ), m_pattern );
15422             case WildcardAtEnd:
15423                 return startsWith( normaliseString( str ), m_pattern );
15424             case WildcardAtBothEnds:
15425                 return contains( normaliseString( str ), m_pattern );
15426             default:
15427                 CATCH_INTERNAL_ERROR( "Unknown enum" );
15428         }
15429     }
15430
15431     std::string WildcardPattern::normaliseString( std::string const& str ) const {
15432         return trim( m_caseSensitivity == CaseSensitive::No ? toLower( str ) : str );
15433     }
15434 }
15435
15436 // end catch_wildcard_pattern.cpp

```

```

15437 // start catch_xmlwriter.cpp
15438
15439 #include <iomanip>
15440 #include <type_traits>
15441
15442 namespace Catch {
15443
15444     namespace {
15445
15446         size_t trailingBytes(unsigned char c) {
15447             if ((c & 0xE0) == 0xC0) {
15448                 return 2;
15449             }
15450             if ((c & 0xF0) == 0xE0) {
15451                 return 3;
15452             }
15453             if ((c & 0xF8) == 0xF0) {
15454                 return 4;
15455             }
15456             CATCH_INTERNAL_ERROR("Invalid multibyte utf-8 start byte encountered");
15457         }
15458
15459         uint32_t headerValue(unsigned char c) {
15460             if ((c & 0xE0) == 0xC0) {
15461                 return c & 0x1F;
15462             }
15463             if ((c & 0xF0) == 0xE0) {
15464                 return c & 0x0F;
15465             }
15466             if ((c & 0xF8) == 0xF0) {
15467                 return c & 0x07;
15468             }
15469             CATCH_INTERNAL_ERROR("Invalid multibyte utf-8 start byte encountered");
15470         }
15471
15472         void hexEscapeChar(std::ostream& os, unsigned char c) {
15473             std::ios_base::fmtflags f(os.flags());
15474             os << "\\x"
15475                 << std::uppercase << std::hex << std::setfill('0') << std::setw(2)
15476                 << static_cast<int>(c);
15477             os.flags(f);
15478         }
15479
15480         bool shouldNewline(XmlFormatting fmt) {
15481             return !(static_cast<std::underlying_type<XmlFormatting>::type>(fmt &
15482                 XmlFormatting::Newline));
15483         }
15484
15485         bool shouldIndent(XmlFormatting fmt) {
15486             return !(static_cast<std::underlying_type<XmlFormatting>::type>(fmt &
15487                 XmlFormatting::Indent));
15488         }
15489     } // anonymous namespace
15490
15491     XmlFormatting operator | (XmlFormatting lhs, XmlFormatting rhs) {
15492         return static_cast<XmlFormatting>(
15493             static_cast<std::underlying_type<XmlFormatting>::type>(lhs) |
15494             static_cast<std::underlying_type<XmlFormatting>::type>(rhs)
15495         );
15496
15497     XmlFormatting operator & (XmlFormatting lhs, XmlFormatting rhs) {
15498         return static_cast<XmlFormatting>(
15499             static_cast<std::underlying_type<XmlFormatting>::type>(lhs) &
15500             static_cast<std::underlying_type<XmlFormatting>::type>(rhs)
15501         );
15502     }
15503
15504     XmlEncode::XmlEncode( std::string const& str, ForWhat forWhat )
15505     :   m_str( str ),
15506         m_forWhat( forWhat )
15507     {}
15508
15509     void XmlEncode::encodeTo( std::ostream& os ) const {
15510         // Apostrophe escaping not necessary if we always use " to write attributes
15511         // (see: http://www.w3.org/TR/xml/#syntax)
15512
15513         for( std::size_t idx = 0; idx < m_str.size(); ++idx ) {
15514             unsigned char c = m_str[idx];
15515             switch (c) {
15516                 case '<': os << "&lt;"; break;
15517                 case '&': os << "&amp;"; break;
15518
15519                 case '>':
15520                     // See: http://www.w3.org/TR/xml/#syntax
15521                     if (idx > 2 && m_str[idx - 1] == '[' && m_str[idx - 2] == '[')

```

```

15522         os « "&gt;";
15523     else
15524         os « c;
15525     break;
15526
15527     case '\\':
15528         if (m_forWhat == ForAttributes)
15529             os « "&quot;";
15530         else
15531             os « c;
15532         break;
15533
15534     default:
15535         // Check for control characters and invalid utf-8
15536
15537         // Escape control characters in standard ascii
15538         // see
15539         http://stackoverflow.com/questions/404107/why-are-control-characters-illegal-in-xml-1-0
15540         if (c < 0x09 || (c > 0x0D && c < 0x20) || c == 0x7F) {
15541             hexEscapeChar(os, c);
15542             break;
15543         }
15544
15545         // Plain ASCII: Write it to stream
15546         if (c < 0x7F) {
15547             os « c;
15548             break;
15549         }
15550
15551         // UTF-8 territory
15552         // Check if the encoding is valid and if it is not, hex escape bytes.
15553         // Important: We do not check the exact decoded values for validity, only the encoding
15554         format
15555         // First check that this bytes is a valid lead byte:
15556         // This means that it is not encoded as 1111 1XXX
15557         // Or as 10XX XXXX
15558         if (c < 0xC0 ||
15559             c >= 0xF8) {
15560             hexEscapeChar(os, c);
15561             break;
15562         }
15563
15564         auto encBytes = trailingBytes(c);
15565         // Are there enough bytes left to avoid accessing out-of-bounds memory?
15566         if (idx + encBytes - 1 >= m_str.size()) {
15567             hexEscapeChar(os, c);
15568             break;
15569         }
15570
15571         // The header is valid, check data
15572         // The next encBytes bytes must together be a valid utf-8
15573         // This means: bitpattern 10XX XXXX and the extracted value is sane (ish)
15574         bool valid = true;
15575         uint32_t value = headerValue(c);
15576         for (std::size_t n = 1; n < encBytes; ++n) {
15577             unsigned char nc = m_str[idx + n];
15578             valid &= ((nc & 0xC0) == 0x80);
15579             value = (value « 6) | (nc & 0x3F);
15580         }
15581
15582         if (
15583             // Wrong bit pattern of following bytes
15584             (!valid) ||
15585             // Overlong encodings
15586             (value < 0x80) ||
15587             (0x80 <= value && value < 0x800 && encBytes > 2) ||
15588             (0x800 < value && value < 0x10000 && encBytes > 3) ||
15589             // Encoded value out of range
15590             (value >= 0x110000)
15591         ) {
15592             hexEscapeChar(os, c);
15593             break;
15594         }
15595
15596         // If we got here, this is in fact a valid(ish) utf-8 sequence
15597         for (std::size_t n = 0; n < encBytes; ++n) {
15598             os « m_str[idx + n];
15599         }
15600         idx += encBytes - 1;
15601         break;
15602     }
15603 }
15604
15605 std::ostream& operator « ( std::ostream& os, XmlEncode const& xmlEncode ) {
15606     xmlEncode.encodeTo( os );
15607     return os;
15608 }

```



```

15607
15608     XmlWriter::ScopedElement::ScopedElement( XmlWriter* writer, XmlFormatting fmt )
15609     :   m_writer( writer ),
15610         m_fmt( fmt )
15611     {}
15612
15613     XmlWriter::ScopedElement::ScopedElement( ScopedElement&& other ) noexcept
15614     :   m_writer( other.m_writer ),
15615         m_fmt( other.m_fmt )
15616     {
15617         other.m_writer = nullptr;
15618         other.m_fmt = XmlFormatting::None;
15619     }
15620     XmlWriter::ScopedElement& XmlWriter::ScopedElement::operator=( ScopedElement&& other ) noexcept {
15621         if ( m_writer ) {
15622             m_writer->endElement();
15623         }
15624         m_writer = other.m_writer;
15625         other.m_writer = nullptr;
15626         m_fmt = other.m_fmt;
15627         other.m_fmt = XmlFormatting::None;
15628         return *this;
15629     }
15630
15631     XmlWriter::ScopedElement::~ScopedElement() {
15632         if (m_writer) {
15633             m_writer->endElement(m_fmt);
15634         }
15635     }
15636
15637     XmlWriter::ScopedElement& XmlWriter::ScopedElement::writeText( std::string const& text,
15638     XmlFormatting fmt ) {
15639         m_writer->writeText( text, fmt );
15640         return *this;
15641     }
15642
15643     XmlWriter::XmlWriter( std::ostream& os ) : m_os( os )
15644     {
15645         writeDeclaration();
15646     }
15647
15648     XmlWriter::~XmlWriter() {
15649         while (!m_tags.empty()) {
15650             endElement();
15651         }
15652         newlineIfNecessary();
15653     }
15654
15655     XmlWriter& XmlWriter::startElement( std::string const& name, XmlFormatting fmt ) {
15656         ensureTagClosed();
15657         newlineIfNecessary();
15658         if (shouldIndent(fmt)) {
15659             m_os << m_indent;
15660             m_indent += " ";
15661         }
15662         m_os << '<' << name;
15663         m_tags.push_back( name );
15664         m_tagIsOpen = true;
15665         applyFormatting( fmt );
15666         return *this;
15667     }
15668
15669     XmlWriter::ScopedElement XmlWriter::scopedElement( std::string const& name, XmlFormatting fmt ) {
15670         ScopedElement scoped( this, fmt );
15671         startElement( name, fmt );
15672         return scoped;
15673     }
15674
15675     XmlWriter& XmlWriter::endElement(XmlFormatting fmt) {
15676         m_indent = m_indent.substr(0, m_indent.size() - 2);
15677
15678         if( m_tagIsOpen ) {
15679             m_os << ">";
15680             m_tagIsOpen = false;
15681         } else {
15682             newlineIfNecessary();
15683             if (shouldIndent(fmt)) {
15684                 m_os << m_indent;
15685             }
15686             m_os << "</" << m_tags.back() << ">";
15687         }
15688         m_os << std::flush;
15689         applyFormatting( fmt );
15690         m_tags.pop_back();
15691         return *this;
15692     }

```

```

15693     XmlWriter& XmlWriter::writeAttribute( std::string const& name, std::string const& attribute ) {
15694         if( !name.empty() && !attribute.empty() )
15695             m_os << ' ' << name << "=" << XmlEncode( attribute, XmlEncode::ForAttributes ) << ' ';
15696         return *this;
15697     }
15698
15699     XmlWriter& XmlWriter::writeAttribute( std::string const& name, bool attribute ) {
15700         m_os << ' ' << name << "=" << ( attribute ? "true" : "false" ) << ' ';
15701         return *this;
15702     }
15703
15704     XmlWriter& XmlWriter::writeText( std::string const& text, XmlFormatting fmt ) {
15705         if( !text.empty() ) {
15706             bool tagWasOpen = m_tagIsOpen;
15707             ensureTagClosed();
15708             if (tagWasOpen && shouldIndent(fmt)) {
15709                 m_os << m_indent;
15710             }
15711             m_os << XmlEncode( text );
15712             applyFormatting(fmt);
15713         }
15714         return *this;
15715     }
15716
15717     XmlWriter& XmlWriter::writeComment( std::string const& text, XmlFormatting fmt ) {
15718         ensureTagClosed();
15719         if (shouldIndent(fmt)) {
15720             m_os << m_indent;
15721         }
15722         m_os << "<!--" << text << "-->";
15723         applyFormatting(fmt);
15724         return *this;
15725     }
15726
15727     void XmlWriter::writeStylesheetRef( std::string const& url ) {
15728         m_os << "<?xml-stylesheet type=\"text/xsl\" href=\"" << url << "\"?>\n";
15729     }
15730
15731     XmlWriter& XmlWriter::writeBlankLine() {
15732         ensureTagClosed();
15733         m_os << '\n';
15734         return *this;
15735     }
15736
15737     void XmlWriter::ensureTagClosed() {
15738         if( m_tagIsOpen ) {
15739             m_os << '>' << std::flush;
15740             newlineIfNecessary();
15741             m_tagIsOpen = false;
15742         }
15743     }
15744
15745     void XmlWriter::applyFormatting(XmlFormatting fmt) {
15746         m_needsNewline = shouldNewline(fmt);
15747     }
15748
15749     void XmlWriter::writeDeclaration() {
15750         m_os << "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n";
15751     }
15752
15753     void XmlWriter::newlineIfNecessary() {
15754         if( m_needsNewline ) {
15755             m_os << std::endl;
15756             m_needsNewline = false;
15757         }
15758     }
15759 }
15760 // end catch_xmlwriter.cpp
15761 // start catch_reporter_bases.cpp
15762
15763 #include <cstring>
15764 #include <cfloat>
15765 #include <cstdio>
15766 #include <cassert>
15767 #include <memory>
15768
15769 namespace Catch {
15770     void prepareExpandedExpression( AssertionResult& result ) {
15771         result.getExpandedExpression();
15772     }
15773
15774     // Because formatting using c++ streams is stateful, drop down to C is required
15775     // Alternatively we could use stringstream, but its performance is... not good.
15776     std::string getFormattedDuration( double duration ) {
15777         // Max exponent + 1 is required to represent the whole part
15778         // + 1 for decimal point
15779         // + 3 for the 3 decimal places

```

```

15780         // + 1 for null terminator
15781         const std::size_t maxDoubleSize = DBL_MAX_10_EXP + 1 + 1 + 3 + 1;
15782         char buffer[maxDoubleSize];
15783
15784         // Save previous errno, to prevent sprintf from overwriting it
15785         ErrnoGuard guard;
15786 #ifdef _MSC_VER
15787         sprintf_s(buffer, "%.3f", duration);
15788 #else
15789         std::sprintf(buffer, "%.3f", duration);
15790 #endif
15791         return std::string(buffer);
15792     }
15793
15794     bool shouldShowDuration( IConfig const& config, double duration ) {
15795         if ( config.showDurations() == ShowDurations::Always ) {
15796             return true;
15797         }
15798         if ( config.showDurations() == ShowDurations::Never ) {
15799             return false;
15800         }
15801         const double min = config.minDuration();
15802         return min >= 0 && duration >= min;
15803     }
15804
15805     std::string serializeFilters( std::vector<std::string> const& container ) {
15806         ReusableStringStream oss;
15807         bool first = true;
15808         for (auto&& filter : container)
15809         {
15810             if (!first)
15811                 oss << ' ';
15812             else
15813                 first = false;
15814
15815             oss << filter;
15816         }
15817         return oss.str();
15818     }
15819
15820     TestEventListenerBase::TestEventListenerBase(ReporterConfig const & _config)
15821         :StreamingReporterBase(_config) {}
15822
15823     std::set<Verbosity> TestEventListenerBase::getSupportedVerbsities() {
15824         return { Verbosity::Quiet, Verbosity::Normal, Verbosity::High };
15825     }
15826
15827     void TestEventListenerBase::assertionStarting(AssertionInfo const &) {}
15828
15829     bool TestEventListenerBase::assertionEnded(AssertionStats const &) {
15830         return false;
15831     }
15832
15833 } // end namespace Catch
15834 // end catch_reporter_bases.cpp
15835 // start catch_reporter_compact.cpp
15836
15837 namespace {
15838
15839 #ifdef CATCH_PLATFORM_MAC
15840     const char* failedString() { return "FAILED"; }
15841     const char* passedString() { return "PASSED"; }
15842 #else
15843     const char* failedString() { return "failed"; }
15844     const char* passedString() { return "passed"; }
15845 #endif
15846
15847     // Colour::LightGrey
15848     Catch::Colour::Code dimColour() { return Catch::Colour::FileName; }
15849
15850     std::string bothOrAll( std::size_t count ) {
15851         return count == 1 ? std::string() :
15852             count == 2 ? "both " : "all ";
15853     }
15854
15855 } // anon namespace
15856
15857 namespace Catch {
15858     namespace {
15859         // Colour, message variants:
15860         // - white: No tests ran.
15861         // - red: Failed [both/all] N test cases, failed [both/all] M assertions.
15862         // - white: Passed [both/all] N test cases (no assertions).
15863         // - red: Failed N tests cases, failed M assertions.
15864         // - green: Passed [both/all] N tests cases with M assertions.
15865         void printTotals(std::ostream& out, const Totals& totals) {
15866             if (totals.testCases.total() == 0) {

```

```

15867         out << "No tests ran.";
15868     } else if (totals.testCases.failed == totals.testCases.total()) {
15869         Colour colour(Colour::ResultError);
15870         const std::string qualify_assertions_failed =
15871             totals.assertions.failed == totals.assertions.total() ?
15872             bothOrAll(totals.assertions.failed) : std::string();
15873         out <<
15874             "Failed " << bothOrAll(totals.testCases.failed)
15875             << pluralise(totals.testCases.failed, "test case") << ", "
15876             << "failed " << qualify_assertions_failed <<
15877             pluralise(totals.assertions.failed, "assertion") << ' ';
15878     } else if (totals.assertions.total() == 0) {
15879         out <<
15880             "Passed " << bothOrAll(totals.testCases.total())
15881             << pluralise(totals.testCases.total(), "test case")
15882             << " (no assertions).";
15883     } else if (totals.assertions.failed) {
15884         Colour colour(Colour::ResultError);
15885         out <<
15886             "Failed " << pluralise(totals.testCases.failed, "test case") << ", "
15887             << "failed " << pluralise(totals.assertions.failed, "assertion") << ' ';
15888     } else {
15889         Colour colour(Colour::ResultSuccess);
15890         out <<
15891             "Passed " << bothOrAll(totals.testCases.passed)
15892             << pluralise(totals.testCases.passed, "test case") <<
15893             " with " << pluralise(totals.assertions.passed, "assertion") << ' ';
15894     }
15895 }
15896
15897 // Implementation of CompactReporter formatting
15898 class AssertionPrinter {
15899 public:
15900     AssertionPrinter& operator= (AssertionPrinter const&) = delete;
15901     AssertionPrinter(AssertionPrinter const&) = delete;
15902     AssertionPrinter(std::ostream& _stream, AssertionStats const& _stats, bool _printInfoMessages)
15903         : stream(_stream)
15904         , result(_stats.assertionResult)
15905         , messages(_stats.infoMessages)
15906         , itMessage(_stats.infoMessages.begin())
15907         , printInfoMessages(_printInfoMessages) {}
15908
15909     void print() {
15910         printSourceInfo();
15911
15912         itMessage = messages.begin();
15913
15914         switch (result.getResultType()) {
15915             case ResultWas::Ok:
15916                 printResultType(Colour::ResultSuccess, passedString());
15917                 printOriginalExpression();
15918                 printReconstructedExpression();
15919                 if (!result.hasExpression())
15920                     printRemainingMessages(Colour::None);
15921                 else
15922                     printRemainingMessages();
15923                 break;
15924             case ResultWas::ExpressionFailed:
15925                 if (result.isOk())
15926                     printResultType(Colour::ResultSuccess, failedString() + std::string(" - but was ok"));
15927                 else
15928                     printResultType(Colour::Error, failedString());
15929                 printOriginalExpression();
15930                 printReconstructedExpression();
15931                 printRemainingMessages();
15932                 break;
15933             case ResultWas::ThrewException:
15934                 printResultType(Colour::Error, failedString());
15935                 printIssue("unexpected exception with message:");
15936                 printMessage();
15937                 printExpressionWas();
15938                 printRemainingMessages();
15939                 break;
15940             case ResultWas::FatalErrorCondition:
15941                 printResultType(Colour::Error, failedString());
15942                 printIssue("fatal error condition with message:");
15943                 printMessage();
15944                 printExpressionWas();
15945                 printRemainingMessages();
15946                 break;
15947             case ResultWas::DidntThrowException:
15948                 printResultType(Colour::Error, failedString());
15949                 printIssue("expected exception, got none");
15950                 printExpressionWas();
15951                 printRemainingMessages();
15952                 break;
15953             case ResultWas::Info:

```

```

15954         printResultType(Colour::None, "info");
15955         printMessage();
15956         printRemainingMessages();
15957         break;
15958     case ResultWas::Warning:
15959         printResultType(Colour::None, "warning");
15960         printMessage();
15961         printRemainingMessages();
15962         break;
15963     case ResultWas::ExplicitFailure:
15964         printResultType(Colour::Error, failedString());
15965         printIssue("explicitly");
15966         printRemainingMessages(Colour::None);
15967         break;
15968         // These cases are here to prevent compiler warnings
15969     case ResultWas::Unknown:
15970     case ResultWas::FailureBit:
15971     case ResultWas::Exception:
15972         printResultType(Colour::Error, "** internal error **");
15973         break;
15974     }
15975 }
15976
15977 private:
15978 void printSourceInfo() const {
15979     Colour colourGuard(Colour::FileName);
15980     stream < result.getSourceInfo() < " ':";
15981 }
15982
15983 void printResultType(Colour::Code colour, std::string const& passOrFail) const {
15984     if (!passOrFail.empty()) {
15985         {
15986             Colour colourGuard(colour);
15987             stream < " ' " < passOrFail;
15988         }
15989         stream < ":'";
15990     }
15991 }
15992
15993 void printIssue(std::string const& issue) const {
15994     stream < " ' " < issue;
15995 }
15996
15997 void printExpressionWas() {
15998     if (result.hasExpression()) {
15999         stream < ":'";
16000         {
16001             Colour colour(dimColour());
16002             stream < " expression was:";
16003         }
16004         printOriginalExpression();
16005     }
16006 }
16007
16008 void printOriginalExpression() const {
16009     if (result.hasExpression()) {
16010         stream < " ' " < result.getExpression();
16011     }
16012 }
16013
16014 void printReconstructedExpression() const {
16015     if (result.hasExpandedExpression()) {
16016         {
16017             Colour colour(dimColour());
16018             stream < " for: ";
16019         }
16020         stream < result.getExpandedExpression();
16021     }
16022 }
16023
16024 void printMessage() {
16025     if (itMessage != messages.end()) {
16026         stream < " ' " < itMessage->message < " \n";
16027         ++itMessage;
16028     }
16029 }
16030
16031 void printRemainingMessages(Colour::Code colour = dimColour()) {
16032     if (itMessage == messages.end())
16033         return;
16034
16035     const auto itEnd = messages.cend();
16036     const auto N = static_cast<std::size_t>(std::distance(itMessage, itEnd));
16037
16038     {
16039         Colour colourGuard(colour);
16040         stream < " with " < pluralise(N, "message") < ":'";

```

```

16041     }
16042
16043     while (itMessage != itEnd) {
16044         // If this assertion is a warning ignore any INFO messages
16045         if (printInfoMessages || itMessage->type != ResultWas::Info) {
16046             printMessage();
16047             if (itMessage != itEnd) {
16048                 Colour colourGuard(dimColour());
16049                 stream << " and";
16050             }
16051             continue;
16052         }
16053         ++itMessage;
16054     }
16055 }
16056
16057 private:
16058     std::ostream& stream;
16059     AssertionResult const& result;
16060     std::vector<MessageInfo> messages;
16061     std::vector<MessageInfo>::const_iterator itMessage;
16062     bool printInfoMessages;
16063 };
16064
16065 } // anon namespace
16066
16067     std::string CompactReporter::getDescription() {
16068         return "Reports test results on a single line, suitable for IDEs";
16069     }
16070
16071     void CompactReporter::noMatchingTestCases( std::string const& spec ) {
16072         stream << "No test cases matched '" << spec << "'\n" << std::endl;
16073     }
16074
16075     void CompactReporter::assertionStarting( AssertionInfo const& ) {}
16076
16077     bool CompactReporter::assertionEnded( AssertionStats const& _assertionStats ) {
16078         AssertionResult const& result = _assertionStats.assertionResult;
16079
16080         bool printInfoMessages = true;
16081
16082         // Drop out if result was successful and we're not printing those
16083         if( !_m_config->includeSuccessfulResults() && result.isOk() ) {
16084             if( result.getResultType() != ResultWas::Warning )
16085                 return false;
16086             printInfoMessages = false;
16087         }
16088
16089         AssertionPrinter printer( stream, _assertionStats, printInfoMessages );
16090         printer.print();
16091
16092         stream << std::endl;
16093         return true;
16094     }
16095
16096     void CompactReporter::sectionEnded(SectionStats const& _sectionStats) {
16097         double dur = _sectionStats.durationInSeconds;
16098         if ( shouldShowDuration( *m_config, dur ) ) {
16099             stream << getFormattedDuration( dur ) << " s: " << _sectionStats.sectionInfo.name <<
16100             std::endl;
16101         }
16102     }
16103
16104     void CompactReporter::testRunEnded( TestRunStats const& _testRunStats ) {
16105         printTotals( stream, _testRunStats.totals );
16106         stream << '\n' << std::endl;
16107         StreamingReporterBase::testRunEnded( _testRunStats );
16108     }
16109
16110     CompactReporter::~CompactReporter() {}
16111
16112     CATCH_REGISTER_REPORTER( "compact", CompactReporter )
16113 } // end namespace Catch
16114 // end catch_reporter_compact.cpp
16115 // start catch_reporter_console.cpp
16116
16117 #include <cstdio>
16118 #include <cstdio>
16119
16120 #if defined(_MSC_VER)
16121 #pragma warning(push)
16122 #pragma warning(disable:4061) // Not all labels are EXPLICITLY handled in switch
16123 // Note that 4062 (not all labels are handled and default is missing) is enabled
16124 #endif
16125
16126 #if defined(__clang__)

```

```

16127 # pragma clang diagnostic push
16128 // For simplicity, benchmarking-only helpers are always enabled
16129 # pragma clang diagnostic ignored "-Wunused-function"
16130 #endif
16131
16132 namespace Catch {
16133 namespace {
16134
16135 // Formatter impl for ConsoleReporter
16136 class ConsoleAssertionPrinter {
16137 public:
16138     ConsoleAssertionPrinter& operator= (ConsoleAssertionPrinter const&) = delete;
16139     ConsoleAssertionPrinter(ConsoleAssertionPrinter const&) = delete;
16140     ConsoleAssertionPrinter(std::ostream& _stream, AssertionStats const& _stats, bool
_printInfoMessages)
16141         : stream(_stream),
16142           stats(_stats),
16143           result(_stats.assertionResult),
16144           colour(Colour::None),
16145           message(result.getMessage()),
16146           messages(_stats.infoMessages),
16147           printInfoMessages(_printInfoMessages) {
16148         switch (result.getResultType()) {
16149             case ResultWas::Ok:
16150                 colour = Colour::Success;
16151                 passOrFail = "PASSED";
16152                 //if( result.hasMessage() )
16153                 if (_stats.infoMessages.size() == 1)
16154                     messageLabel = "with message";
16155                 if (_stats.infoMessages.size() > 1)
16156                     messageLabel = "with messages";
16157                 break;
16158             case ResultWas::ExpressionFailed:
16159                 if (result.isOk()) {
16160                     colour = Colour::Success;
16161                     passOrFail = "FAILED - but was ok";
16162                 } else {
16163                     colour = Colour::Error;
16164                     passOrFail = "FAILED";
16165                 }
16166                 if (_stats.infoMessages.size() == 1)
16167                     messageLabel = "with message";
16168                 if (_stats.infoMessages.size() > 1)
16169                     messageLabel = "with messages";
16170                 break;
16171             case ResultWas::ThrewException:
16172                 colour = Colour::Error;
16173                 passOrFail = "FAILED";
16174                 messageLabel = "due to unexpected exception with ";
16175                 if (_stats.infoMessages.size() == 1)
16176                     messageLabel += "message";
16177                 if (_stats.infoMessages.size() > 1)
16178                     messageLabel += "messages";
16179                 break;
16180             case ResultWas::FatalErrorCondition:
16181                 colour = Colour::Error;
16182                 passOrFail = "FAILED";
16183                 messageLabel = "due to a fatal error condition";
16184                 break;
16185             case ResultWas::DidntThrowException:
16186                 colour = Colour::Error;
16187                 passOrFail = "FAILED";
16188                 messageLabel = "because no exception was thrown where one was expected";
16189                 break;
16190             case ResultWas::Info:
16191                 messageLabel = "info";
16192                 break;
16193             case ResultWas::Warning:
16194                 messageLabel = "warning";
16195                 break;
16196             case ResultWas::ExplicitFailure:
16197                 passOrFail = "FAILED";
16198                 colour = Colour::Error;
16199                 if (_stats.infoMessages.size() == 1)
16200                     messageLabel = "explicitly with message";
16201                 if (_stats.infoMessages.size() > 1)
16202                     messageLabel = "explicitly with messages";
16203                 break;
16204             // These cases are here to prevent compiler warnings
16205             case ResultWas::Unknown:
16206             case ResultWas::FailureBit:
16207             case ResultWas::Exception:
16208                 passOrFail = "** internal error **";
16209                 colour = Colour::Error;
16210                 break;
16211         }
16212     }

```

```

16213     }
16214
16215     void print() const {
16216         printSourceInfo();
16217         if (stats.totals.assertions.total() > 0) {
16218             printResultType();
16219             printOriginalExpression();
16220             printReconstructedExpression();
16221         } else {
16222             stream << '\n';
16223         }
16224         printMessage();
16225     }
16226
16227 private:
16228     void printResultType() const {
16229         if (!passOrFail.empty()) {
16230             Colour colourGuard(colour);
16231             stream << passOrFail << ":\n";
16232         }
16233     }
16234     void printOriginalExpression() const {
16235         if (result.hasExpression()) {
16236             Colour colourGuard(Colour::OriginalExpression);
16237             stream << " ";
16238             stream << result.getExpressionInMacro();
16239             stream << '\n';
16240         }
16241     }
16242     void printReconstructedExpression() const {
16243         if (result.hasExpandedExpression()) {
16244             stream << "with expansion:\n";
16245             Colour colourGuard(Colour::ReconstructedExpression);
16246             stream << Column(result.getExpandedExpression()).indent(2) << '\n';
16247         }
16248     }
16249     void printMessage() const {
16250         if (!messageLabel.empty())
16251             stream << messageLabel << ':' << '\n';
16252         for (auto const& msg : messages) {
16253             // If this assertion is a warning ignore any INFO messages
16254             if (printInfoMessages || msg.type != ResultWas::Info)
16255                 stream << Column(msg.message).indent(2) << '\n';
16256         }
16257     }
16258     void printSourceInfo() const {
16259         Colour colourGuard(Colour::FileName);
16260         stream << result.getSourceInfo() << ": ";
16261     }
16262
16263     std::ostream& stream;
16264     AssertionStats const& stats;
16265     AssertionResult const& result;
16266     Colour::Code colour;
16267     std::string passOrFail;
16268     std::string messageLabel;
16269     std::string message;
16270     std::vector<MessageInfo> messages;
16271     bool printInfoMessages;
16272 };
16273
16274 std::size_t makeRatio(std::size_t number, std::size_t total) {
16275     std::size_t ratio = total > 0 ? CATCH_CONFIG_CONSOLE_WIDTH * number / total : 0;
16276     return (ratio == 0 && number > 0) ? 1 : ratio;
16277 }
16278
16279 std::size_t& findMax(std::size_t& i, std::size_t& j, std::size_t& k) {
16280     if (i > j && i > k)
16281         return i;
16282     else if (j > k)
16283         return j;
16284     else
16285         return k;
16286 }
16287
16288 struct ColumnInfo {
16289     enum Justification { Left, Right };
16290     std::string name;
16291     int width;
16292     Justification justification;
16293 };
16294 struct ColumnBreak {};
16295 struct RowBreak {};
16296
16297 class Duration {
16298     enum class Unit {
16299         Auto,

```



```

16300         Nanoseconds,
16301         Microseconds,
16302         Milliseconds,
16303         Seconds,
16304         Minutes
16305     };
16306     static const uint64_t s_nanosecondsInAMicrosecond = 1000;
16307     static const uint64_t s_nanosecondsInAMillisecond = 1000 * s_nanosecondsInAMicrosecond;
16308     static const uint64_t s_nanosecondsInASecond = 1000 * s_nanosecondsInAMillisecond;
16309     static const uint64_t s_nanosecondsInAMinute = 60 * s_nanosecondsInASecond;
16310
16311     double m_inNanoseconds;
16312     Unit m_units;
16313
16314 public:
16315     explicit Duration(double inNanoseconds, Unit units = Unit::Auto)
16316         : m_inNanoseconds(inNanoseconds),
16317         m_units(units) {
16318         if (m_units == Unit::Auto) {
16319             if (m_inNanoseconds < s_nanosecondsInAMicrosecond)
16320                 m_units = Unit::Nanoseconds;
16321             else if (m_inNanoseconds < s_nanosecondsInAMillisecond)
16322                 m_units = Unit::Microseconds;
16323             else if (m_inNanoseconds < s_nanosecondsInASecond)
16324                 m_units = Unit::Milliseconds;
16325             else if (m_inNanoseconds < s_nanosecondsInAMinute)
16326                 m_units = Unit::Seconds;
16327             else
16328                 m_units = Unit::Minutes;
16329         }
16330     }
16331
16332     auto value() const -> double {
16333         switch (m_units) {
16334             case Unit::Microseconds:
16335                 return m_inNanoseconds / static_cast<double>(s_nanosecondsInAMicrosecond);
16336             case Unit::Milliseconds:
16337                 return m_inNanoseconds / static_cast<double>(s_nanosecondsInAMillisecond);
16338             case Unit::Seconds:
16339                 return m_inNanoseconds / static_cast<double>(s_nanosecondsInASecond);
16340             case Unit::Minutes:
16341                 return m_inNanoseconds / static_cast<double>(s_nanosecondsInAMinute);
16342             default:
16343                 return m_inNanoseconds;
16344         }
16345     }
16346
16347     auto unitsAsString() const -> std::string {
16348         switch (m_units) {
16349             case Unit::Nanoseconds:
16350                 return "ns";
16351             case Unit::Microseconds:
16352                 return "us";
16353             case Unit::Milliseconds:
16354                 return "ms";
16355             case Unit::Seconds:
16356                 return "s";
16357             case Unit::Minutes:
16358                 return "m";
16359             default:
16360                 return "** internal error **";
16361         }
16362     }
16363
16364     friend auto operator << (std::ostream& os, Duration const& duration) -> std::ostream& {
16365         return os << duration.value() << ' ' << duration.unitsAsString();
16366     }
16367 };
16368 } // end anon namespace
16369
16370 class TablePrinter {
16371     std::ostream& m_os;
16372     std::vector<ColumnInfo> m_columnInfos;
16373     std::ostream& m_oss;
16374     int m_currentColumn = -1;
16375     bool m_isOpen = false;
16376
16377 public:
16378     TablePrinter( std::ostream& os, std::vector<ColumnInfo> columnInfos )
16379         : m_os( os ),
16380         m_columnInfos( std::move( columnInfos ) ) {}
16381
16382     auto columnInfos() const -> std::vector<ColumnInfo> const& {
16383         return m_columnInfos;
16384     }
16385
16386     void open() {

```

```

16387         if (!m_isOpen) {
16388             m_isOpen = true;
16389             *this « RowBreak();
16390
16391             Columns headerCols;
16392             Spacer spacer(2);
16393             for (auto const& info : m_columnInfos) {
16394                 headerCols += Column(info.name).width(static_cast<std::size_t>(info.width - 2));
16395                 headerCols += spacer;
16396             }
16397             m_os « headerCols « '\n';
16398
16399             m_os « Catch::getLineOfChars<'-'>() « '\n';
16400         }
16401     }
16402     void close() {
16403         if (m_isOpen) {
16404             *this « RowBreak();
16405             m_os « std::endl;
16406             m_isOpen = false;
16407         }
16408     }
16409
16410     template<typename T>
16411     friend TablePrinter& operator « (TablePrinter& tp, T const& value) {
16412         tp.m_oss « value;
16413         return tp;
16414     }
16415
16416     friend TablePrinter& operator « (TablePrinter& tp, ColumnBreak) {
16417         auto colStr = tp.m_oss.str();
16418         const auto strSize = colStr.size();
16419         tp.m_oss.str("");
16420         tp.open();
16421         if (tp.m_currentColumn == static_cast<int>(tp.m_columnInfos.size() - 1)) {
16422             tp.m_currentColumn = -1;
16423             tp.m_os « '\n';
16424         }
16425         tp.m_currentColumn++;
16426
16427         auto colInfo = tp.m_columnInfos[tp.m_currentColumn];
16428         auto padding = (strSize + 1 < static_cast<std::size_t>(colInfo.width))
16429             ? std::string(colInfo.width - (strSize + 1), ' ')
16430             : std::string();
16431         if (colInfo.justification == ColumnInfo::Left)
16432             tp.m_os « colStr « padding « ' ';
16433         else
16434             tp.m_os « padding « colStr « ' ';
16435         return tp;
16436     }
16437
16438     friend TablePrinter& operator « (TablePrinter& tp, RowBreak) {
16439         if (tp.m_currentColumn > 0) {
16440             tp.m_os « '\n';
16441             tp.m_currentColumn = -1;
16442         }
16443         return tp;
16444     }
16445 };
16446
16447 ConsoleReporter::ConsoleReporter(ReporterConfig const& config)
16448 : StreamingReporterBase(config),
16449   m_tablePrinter(new TablePrinter(config.stream(),
16450   [&config]() -> std::vector<ColumnInfo> {
16451       if (config.fullConfig()->benchmarkNoAnalysis())
16452       {
16453           return{
16454               { "benchmark name", CATCH_CONFIG_CONSOLE_WIDTH - 43, ColumnInfo::Left },
16455               { "      samples", 14, ColumnInfo::Right },
16456               { "   iterations", 14, ColumnInfo::Right },
16457               { "           mean", 14, ColumnInfo::Right }
16458           };
16459       }
16460       else
16461       {
16462           return{
16463               { "benchmark name", CATCH_CONFIG_CONSOLE_WIDTH - 43, ColumnInfo::Left },
16464               { "samples      mean      std dev", 14, ColumnInfo::Right },
16465               { "iterations  low mean   low std dev", 14, ColumnInfo::Right },
16466               { "estimated   high mean  high std dev", 14, ColumnInfo::Right }
16467           };
16468       }
16469   }())) {}
16470 ConsoleReporter::~ConsoleReporter() = default;
16471
16472 std::string ConsoleReporter::getDescription() {
16473     return "Reports test results as plain lines of text";

```

```

16474 }
16475
16476 void ConsoleReporter::noMatchingTestCases(std::string const& spec) {
16477     stream << "No test cases matched '" << spec << "'\n" << std::endl;
16478 }
16479
16480 void ConsoleReporter::reportInvalidArguments(std::string const& arg){
16481     stream << "Invalid Filter: " << arg << std::endl;
16482 }
16483
16484 void ConsoleReporter::assertionStarting(AssertionInfo const&) {}
16485
16486 bool ConsoleReporter::assertionEnded(AssertionStats const& _assertionStats) {
16487     AssertionResult const& result = _assertionStats.assertionResult;
16488
16489     bool includeResults = m_config->includeSuccessfulResults() || !result.isOk();
16490
16491     // Drop out if result was successful but we're not printing them.
16492     if (!includeResults && result.getResultType() != ResultWas::Warning)
16493         return false;
16494
16495     lazyPrint();
16496
16497     ConsoleAssertionPrinter printer(stream, _assertionStats, includeResults);
16498     printer.print();
16499     stream << std::endl;
16500     return true;
16501 }
16502
16503 void ConsoleReporter::sectionStarting(SectionInfo const& _sectionInfo) {
16504     m_tablePrinter->close();
16505     m_headerPrinted = false;
16506     StreamingReporterBase::sectionStarting(_sectionInfo);
16507 }
16508
16509 void ConsoleReporter::sectionEnded(SectionStats const& _sectionStats) {
16510     m_tablePrinter->close();
16511     if (_sectionStats.missingAssertions) {
16512         lazyPrint();
16513         Colour colour(Colour::ResultError);
16514         if (m_sectionStack.size() > 1)
16515             stream << "\nNo assertions in section";
16516         else
16517             stream << "\nNo assertions in test case";
16518         stream << "' '" << _sectionStats.sectionInfo.name << "'\n" << std::endl;
16519     }
16520     double dur = _sectionStats.durationInSeconds;
16521     if (shouldShowDuration(*m_config, dur)) {
16522         stream << getFormattedDuration(dur) << " s: " << _sectionStats.sectionInfo.name << std::endl;
16523     }
16524     if (m_headerPrinted) {
16525         m_headerPrinted = false;
16526     }
16527     StreamingReporterBase::sectionEnded(_sectionStats);
16528 }
16529 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
16530 void ConsoleReporter::benchmarkPreparing(std::string const& name) {
16531     lazyPrintWithoutClosingBenchmarkTable();
16532
16533     auto nameCol = Column(name).width(static_cast<std::size_t>(m_tablePrinter->columnInfos()[0].width
16534 - 2));
16535
16536     bool firstLine = true;
16537     for (auto line : nameCol) {
16538         if (!firstLine)
16539             (*m_tablePrinter) << ColumnBreak() << ColumnBreak() << ColumnBreak();
16540         else
16541             firstLine = false;
16542         (*m_tablePrinter) << line << ColumnBreak();
16543     }
16544 }
16545
16546 void ConsoleReporter::benchmarkStarting(BenchmarkInfo const& info) {
16547     (*m_tablePrinter) << info.samples << ColumnBreak()
16548     << info.iterations << ColumnBreak();
16549     if (!m_config->benchmarkNoAnalysis())
16550         (*m_tablePrinter) << Duration(info.estimatedDuration) << ColumnBreak();
16551 }
16552
16553 void ConsoleReporter::benchmarkEnded(BenchmarkStats<> const& stats) {
16554     if (m_config->benchmarkNoAnalysis())
16555     {
16556         (*m_tablePrinter) << Duration(stats.mean.point.count()) << ColumnBreak();
16557     }
16558     else
16559     {
16560         (*m_tablePrinter) << ColumnBreak()

```

```

16560         « Duration(stats.mean.point.count()) « ColumnBreak()
16561         « Duration(stats.mean.lower_bound.count()) « ColumnBreak()
16562         « Duration(stats.mean.upper_bound.count()) « ColumnBreak() « ColumnBreak()
16563         « Duration(stats.standardDeviation.point.count()) « ColumnBreak()
16564         « Duration(stats.standardDeviation.lower_bound.count()) « ColumnBreak()
16565         « Duration(stats.standardDeviation.upper_bound.count()) « ColumnBreak() « ColumnBreak() «
ColumnBreak() « ColumnBreak() « ColumnBreak();
16566     }
16567 }
16568
16569 void ConsoleReporter::benchmarkFailed(std::string const& error) {
16570     Colour colour(Colour::Red);
16571     (*m_tablePrinter)
16572         « "Benchmark failed (" « error « ') '
16573         « ColumnBreak() « RowBreak();
16574 }
16575 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
16576
16577 void ConsoleReporter::testCaseEnded(TestCaseStats const& _testCaseStats) {
16578     m_tablePrinter->close();
16579     StreamingReporterBase::testCaseEnded(_testCaseStats);
16580     m_headerPrinted = false;
16581 }
16582 void ConsoleReporter::testGroupEnded(TestGroupStats const& _testGroupStats) {
16583     if (currentGroupInfo.used) {
16584         printSummaryDivider();
16585         stream « "Summary for group '" « _testGroupStats.groupInfo.name « "':\n";
16586         printTotals(_testGroupStats.totals);
16587         stream « '\n' « std::endl;
16588     }
16589     StreamingReporterBase::testGroupEnded(_testGroupStats);
16590 }
16591 void ConsoleReporter::testRunEnded(TestRunStats const& _testRunStats) {
16592     printTotalsDivider(_testRunStats.totals);
16593     printTotals(_testRunStats.totals);
16594     stream « std::endl;
16595     StreamingReporterBase::testRunEnded(_testRunStats);
16596 }
16597 void ConsoleReporter::testRunStarting(TestRunInfo const& _testInfo) {
16598     StreamingReporterBase::testRunStarting(_testInfo);
16599     printTestFilters();
16600 }
16601
16602 void ConsoleReporter::lazyPrint() {
16603     m_tablePrinter->close();
16604     lazyPrintWithoutClosingBenchmarkTable();
16605 }
16606
16607 void ConsoleReporter::lazyPrintWithoutClosingBenchmarkTable() {
16608     if (!currentTestRunInfo.used)
16609         lazyPrintRunInfo();
16610     if (!currentGroupInfo.used)
16611         lazyPrintGroupInfo();
16612     if (!m_headerPrinted) {
16613         printTestCaseAndSectionHeader();
16614         m_headerPrinted = true;
16615     }
16616 }
16617 void ConsoleReporter::lazyPrintRunInfo() {
16618     stream « '\n' « getLineOfChars<'~'>() « '\n';
16619     Colour colour(Colour::SecondaryText);
16620     stream « currentTestRunInfo->name
16621         « " is a Catch v" « libraryVersion() « " host application.\n"
16622         « "Run with -? for options\n\n";
16623     if (m_config->rngSeed() != 0)
16624         stream « "Randomness seeded to: " « m_config->rngSeed() « "\n\n";
16625     currentTestRunInfo.used = true;
16626 }
16627 void ConsoleReporter::lazyPrintGroupInfo() {
16628     if (!currentGroupInfo->name.empty() && currentGroupInfo->groupsCounts > 1) {
16629         printClosedHeader("Group: " + currentGroupInfo->name);
16630         currentGroupInfo.used = true;
16631     }
16632 }
16633 void ConsoleReporter::printTestCaseAndSectionHeader() {
16634     assert(!m_sectionStack.empty());
16635     printOpenHeader(currentTestCaseInfo->name);
16636
16637     if (m_sectionStack.size() > 1) {
16638         Colour colourGuard(Colour::Headers);
16639         auto

```

```

16646         it = m_sectionStack.begin() + 1, // Skip first section (test case)
16647         itEnd = m_sectionStack.end();
16648     for (; it != itEnd; ++it)
16649         printHeaderString(it->name, 2);
16650 }
16651
16652 SourceLineInfo lineInfo = m_sectionStack.back().lineInfo;
16653
16654 stream << getLineOfChars<'-'>() << '\n';
16655 Colour colourGuard(Colour::FileName);
16656 stream << lineInfo << '\n';
16657 stream << getLineOfChars<'.'>() << '\n' << std::endl;
16658 }
16659
16660 void ConsoleReporter::printClosedHeader(std::string const& _name) {
16661     printOpenHeader(_name);
16662     stream << getLineOfChars<'.'>() << '\n';
16663 }
16664 void ConsoleReporter::printOpenHeader(std::string const& _name) {
16665     stream << getLineOfChars<'-'>() << '\n';
16666     {
16667         Colour colourGuard(Colour::Headers);
16668         printHeaderString(_name);
16669     }
16670 }
16671
16672 // if string has a : in first line will set indent to follow it on
16673 // subsequent lines
16674 void ConsoleReporter::printHeaderString(std::string const& _string, std::size_t indent) {
16675     std::size_t i = _string.find(": ");
16676     if (i != std::string::npos)
16677         i += 2;
16678     else
16679         i = 0;
16680     stream << Column(_string).indent(indent + i).initialIndent(indent) << '\n';
16681 }
16682
16683 struct SummaryColumn {
16684     SummaryColumn( std::string _label, Colour::Code _colour )
16685     :   label( std::move( _label ) ),
16686         colour( _colour ) {}
16687     SummaryColumn addRow( std::size_t count ) {
16688         ReusableStringStream rss;
16689         rss << count;
16690         std::string row = rss.str();
16691         for (auto& oldRow : rows) {
16692             while (oldRow.size() < row.size())
16693                 oldRow = ' ' + oldRow;
16694             while (oldRow.size() > row.size())
16695                 row = ' ' + row;
16696         }
16697         rows.push_back(row);
16698         return *this;
16699     }
16700     std::string label;
16701     Colour::Code colour;
16702     std::vector<std::string> rows;
16703 };
16704
16705 void ConsoleReporter::printTotals( Totals const& totals ) {
16706     if (totals.testCases.total() == 0) {
16707         stream << Colour(Colour::Warning) << "No tests ran\n";
16708     } else if (totals.assertions.total() > 0 && totals.testCases.allPassed()) {
16709         stream << Colour(Colour::ResultSuccess) << "All tests passed";
16710         stream << " ("
16711             << pluralise(totals.assertions.passed, "assertion") << " in "
16712             << pluralise(totals.testCases.passed, "test case") << ') '
16713             << '\n';
16714     } else {
16715         std::vector<SummaryColumn> columns;
16716         columns.push_back(SummaryColumn("", Colour::None)
16717             .addRow(totals.testCases.total())
16718             .addRow(totals.assertions.total()));
16719         columns.push_back(SummaryColumn("passed", Colour::Success)
16720             .addRow(totals.testCases.passed)
16721             .addRow(totals.assertions.passed));
16722         columns.push_back(SummaryColumn("failed", Colour::ResultError)
16723             .addRow(totals.testCases.failed)
16724             .addRow(totals.assertions.failed));
16725         columns.push_back(SummaryColumn("failed as expected", Colour::ResultExpectedFailure)
16726             .addRow(totals.testCases.failedButOk)
16727             .addRow(totals.assertions.failedButOk));
16728     }
16729 }

```

```

16733         printSummaryRow("test cases", columns, 0);
16734         printSummaryRow("assertions", columns, 1);
16735     }
16736 }
16737 void ConsoleReporter::printSummaryRow(std::string const& label, std::vector<SummaryColumn> const&
cols, std::size_t row) {
16738     for (auto col : cols) {
16739         std::string value = col.rows[row];
16740         if (col.label.empty()) {
16741             stream << label << ": ";
16742             if (value != "0")
16743                 stream << value;
16744             else
16745                 stream << Colour(Colour::Warning) << "- none -";
16746         } else if (value != "0") {
16747             stream << Colour(Colour::LightGrey) << " | ";
16748             stream << Colour(col.colour)
16749                 << value << ' ' << col.label;
16750         }
16751     }
16752     stream << '\n';
16753 }
16754
16755 void ConsoleReporter::printTotalsDivider(Totals const& totals) {
16756     if (totals.testCases.total() > 0) {
16757         std::size_t failedRatio = makeRatio(totals.testCases.failed, totals.testCases.total());
16758         std::size_t failedButOkRatio = makeRatio(totals.testCases.failedButOk,
totals.testCases.total());
16759         std::size_t passedRatio = makeRatio(totals.testCases.passed, totals.testCases.total());
16760         while (failedRatio + failedButOkRatio + passedRatio < CATCH_CONFIG_CONSOLE_WIDTH - 1)
16761             findMax(failedRatio, failedButOkRatio, passedRatio)++;
16762         while (failedRatio + failedButOkRatio + passedRatio > CATCH_CONFIG_CONSOLE_WIDTH - 1)
16763             findMax(failedRatio, failedButOkRatio, passedRatio)--;
16764
16765         stream << Colour(Colour::Error) << std::string(failedRatio, '=');
16766         stream << Colour(Colour::ResultExpectedFailure) << std::string(failedButOkRatio, '=');
16767         if (totals.testCases.allPassed())
16768             stream << Colour(Colour::ResultSuccess) << std::string(passedRatio, '=');
16769         else
16770             stream << Colour(Colour::Success) << std::string(passedRatio, '=');
16771     } else {
16772         stream << Colour(Colour::Warning) << std::string(CATCH_CONFIG_CONSOLE_WIDTH - 1, '=');
16773     }
16774     stream << '\n';
16775 }
16776 void ConsoleReporter::printSummaryDivider() {
16777     stream << getLineOfChars<'-'>() << '\n';
16778 }
16779
16780 void ConsoleReporter::printTestFilters() {
16781     if (m_config->testSpec().hasFilters()) {
16782         Colour guard(Colour::BrightYellow);
16783         stream << "Filters: " << serializeFilters(m_config->getTestsOrTags()) << '\n';
16784     }
16785 }
16786
16787 CATCH_REGISTER_REPORTER("console", ConsoleReporter)
16788
16789 } // end namespace Catch
16790
16791 #if defined(_MSC_VER)
16792 #pragma warning(pop)
16793 #endif
16794
16795 #if defined(__clang__)
16796 #pragma clang diagnostic pop
16797 #endif
16798 // end catch_reporter_console.cpp
16799 // start catch_reporter_junit.cpp
16800
16801 #include <cassert>
16802 #include <sstream>
16803 #include <ctime>
16804 #include <algorithm>
16805 #include <iomanip>
16806
16807 namespace Catch {
16808     namespace {
16809         std::string getCurrentTimestamp() {
16810             // Beware, this is not reentrant because of backward compatibility issues
16811             // Also, UTC only, again because of backward compatibility (%z is C++11)
16812             time_t rawtime;
16813             std::time(&rawtime);
16814             auto const timeStampSize = sizeof("2017-01-16T17:06:45Z");
16815             #ifndef _MSC_VER
16816

```

```

16818         std::tm timeInfo = {};
16819         gmtime_s(&timeInfo, &rawtime);
16820     #else
16821         std::tm* timeInfo;
16822         timeInfo = std::gmtime(&rawtime);
16823     #endif
16824
16825     char timeStamp[timeStampSize];
16826     const char * const fmt = "%Y-%m-%dT%H:%M:%SZ";
16827
16828     #ifdef _MSC_VER
16829         std::strftime(timeStamp, timeStampSize, fmt, &timeInfo);
16830     #else
16831         std::strftime(timeStamp, timeStampSize, fmt, timeInfo);
16832     #endif
16833     return std::string(timeStamp, timeStampSize-1);
16834 }
16835
16836 std::string fileNameTag(const std::vector<std::string> &tags) {
16837     auto it = std::find_if(begin(tags),
16838                             end(tags),
16839                             [] (std::string const& tag) {return tag.front() == '#'; });
16840     if (it != tags.end())
16841         return it->substr(1);
16842     return std::string();
16843 }
16844
16845 // Formats the duration in seconds to 3 decimal places.
16846 // This is done because some genius defined Maven Surefire schema
16847 // in a way that only accepts 3 decimal places, and tools like
16848 // Jenkins use that schema for validation JUnit reporter output.
16849 std::string formatDuration( double seconds ) {
16850     ReusableStringStream rss;
16851     rss << std::fixed << std::setprecision( 3 ) << seconds;
16852     return rss.str();
16853 }
16854
16855 } // anonymous namespace
16856
16857 JUnitReporter::JUnitReporter( ReporterConfig const& _config )
16858 :   CumulativeReporterBase( _config ),
16859     xml( _config.stream() )
16860 {
16861     m_reporterPrefs.shouldRedirectStdOut = true;
16862     m_reporterPrefs.shouldReportAllAssertions = true;
16863 }
16864
16865 JUnitReporter::~JUnitReporter() {}
16866
16867 std::string JUnitReporter::getDescription() {
16868     return "Reports test results in an XML format that looks like Ant's junitreport target";
16869 }
16870
16871 void JUnitReporter::noMatchingTestCases( std::string const& /*spec*/ ) {}
16872
16873 void JUnitReporter::testRunStarting( TestRunInfo const& runInfo ) {
16874     CumulativeReporterBase::testRunStarting( runInfo );
16875     xml.startElement( "testsuites" );
16876 }
16877
16878 void JUnitReporter::testGroupStarting( GroupInfo const& groupInfo ) {
16879     suiteTimer.start();
16880     stdOutForSuite.clear();
16881     stdErrForSuite.clear();
16882     unexpectedExceptions = 0;
16883     CumulativeReporterBase::testGroupStarting( groupInfo );
16884 }
16885
16886 void JUnitReporter::testCaseStarting( TestCaseInfo const& testCaseInfo ) {
16887     m_okToFail = testCaseInfo.okToFail();
16888 }
16889
16890 bool JUnitReporter::assertionEnded( AssertionStats const& assertionStats ) {
16891     if( assertionStats.assertionResult.getResultType() == ResultWas::ThrewException && !m_okToFail )
16892         unexpectedExceptions++;
16893     return CumulativeReporterBase::assertionEnded( assertionStats );
16894 }
16895
16896 void JUnitReporter::testCaseEnded( TestCaseStats const& testCaseStats ) {
16897     stdOutForSuite += testCaseStats.stdOut;
16898     stdErrForSuite += testCaseStats.stdErr;
16899     CumulativeReporterBase::testCaseEnded( testCaseStats );
16900 }
16901
16902 void JUnitReporter::testGroupEnded( TestGroupStats const& testGroupStats ) {
16903     double suiteTime = suiteTimer.getElapsedSeconds();

```

```

16904         CumulativeReporterBase::testGroupEnded( testGroupStats );
16905         writeGroup( *m_testGroups.back(), suiteTime );
16906     }
16907
16908     void JunitReporter::testRunEndedCumulative() {
16909         xml.endElement();
16910     }
16911
16912     void JunitReporter::writeGroup( TestGroupNode const& groupNode, double suiteTime ) {
16913         XmlWriter::ScopedElement e = xml.scopedElement( "testsuite" );
16914
16915         TestGroupStats const& stats = groupNode.value;
16916         xml.writeAttribute( "name", stats.groupInfo.name );
16917         xml.writeAttribute( "errors", unexpectedExceptions );
16918         xml.writeAttribute( "failures", stats.totals.assertions.failed-unexpectedExceptions );
16919         xml.writeAttribute( "tests", stats.totals.assertions.total() );
16920         xml.writeAttribute( "hostname", "tbd" ); // !TBD
16921         if( m_config->showDurations() == ShowDurations::Never )
16922             xml.writeAttribute( "time", "" );
16923         else
16924             xml.writeAttribute( "time", formatDuration( suiteTime ) );
16925         xml.writeAttribute( "timestamp", getCurrentTimestamp() );
16926
16927         // Write properties if there are any
16928         if (m_config->hasTestFilters() || m_config->rngSeed() != 0) {
16929             auto properties = xml.scopedElement("properties");
16930             if (m_config->hasTestFilters()) {
16931                 xml.scopedElement("property")
16932                     .writeAttribute("name", "filters")
16933                     .writeAttribute("value", serializeFilters(m_config->getTestsOrTags()));
16934             }
16935             if (m_config->rngSeed() != 0) {
16936                 xml.scopedElement("property")
16937                     .writeAttribute("name", "random-seed")
16938                     .writeAttribute("value", m_config->rngSeed());
16939             }
16940         }
16941
16942         // Write test cases
16943         for( auto const& child : groupNode.children )
16944             writeTestCase( *child );
16945
16946         xml.scopedElement( "system-out" ).writeText( trim( stdOutForSuite ), XmlFormatting::Newline );
16947         xml.scopedElement( "system-err" ).writeText( trim( stdErrForSuite ), XmlFormatting::Newline );
16948     }
16949
16950     void JunitReporter::writeTestCase( TestCaseNode const& testCaseNode ) {
16951         TestCaseStats const& stats = testCaseNode.value;
16952
16953         // All test cases have exactly one section - which represents the
16954         // test case itself. That section may have 0-n nested sections
16955         assert( testCaseNode.children.size() == 1 );
16956         SectionNode const& rootSection = *testCaseNode.children.front();
16957
16958         std::string className = stats.testInfo.className;
16959
16960         if( className.empty() ) {
16961             className = fileNameTag(stats.testInfo.tags);
16962             if ( className.empty() )
16963                 className = "global";
16964         }
16965
16966         if ( !m_config->name().empty() )
16967             className = m_config->name() + "." + className;
16968
16969         writeSection( className, "", rootSection, stats.testInfo.okToFail() );
16970     }
16971
16972     void JunitReporter::writeSection( std::string const& className,
16973                                     std::string const& rootName,
16974                                     SectionNode const& sectionNode,
16975                                     bool testOkToFail ) {
16976         std::string name = trim( sectionNode.stats.sectionInfo.name );
16977         if( !rootName.empty() )
16978             name = rootName + '/' + name;
16979
16980         if( !sectionNode.assertions.empty() ||
16981             !sectionNode.stdOut.empty() ||
16982             !sectionNode.stdErr.empty() ) {
16983             XmlWriter::ScopedElement e = xml.scopedElement( "testcase" );
16984             if( className.empty() ) {
16985                 xml.writeAttribute( "classname", name );
16986                 xml.writeAttribute( "name", "root" );
16987             }
16988             else {
16989                 xml.writeAttribute( "classname", className );
16990                 xml.writeAttribute( "name", name );

```



```

16991         }
16992         xml.writeAttribute( "time", formatDuration( sectionNode.stats.durationInSeconds ) );
16993         // This is not ideal, but it should be enough to mimic gtest's
16994         // junit output.
16995         // Ideally the JUnit reporter would also handle `skipTest`
16996         // events and write those out appropriately.
16997         xml.writeAttribute( "status", "run" );
16998
16999         if (sectionNode.stats.assertions.failedButOk) {
17000             xml.scopedElement("skipped")
17001                 .writeAttribute("message", "TEST_CASE tagged with !mayfail");
17002         }
17003
17004         writeAssertions( sectionNode );
17005
17006         if( !sectionNode.stdOut.empty() )
17007             XmlFormatting::Newline ;
17008         if( !sectionNode.stdErr.empty() )
17009             XmlFormatting::Newline ;
17010     }
17011     for( auto const& childNode : sectionNode.childSections )
17012         if( className.empty() )
17013             writeSection( name, "", *childNode, testOkToFail );
17014     else
17015         writeSection( className, name, *childNode, testOkToFail );
17016 }
17017
17018 void JunitReporter::writeAssertions( SectionNode const& sectionNode ) {
17019     for( auto const& assertion : sectionNode.assertions )
17020         writeAssertion( assertion );
17021 }
17022
17023 void JunitReporter::writeAssertion( AssertionStats const& stats ) {
17024     AssertionResult const& result = stats.assertionResult;
17025     if( !result.isOk() ) {
17026         std::string elementName;
17027         switch( result.getResultType() ) {
17028             case ResultWas::ThrowException:
17029             case ResultWas::FatalErrorCondition:
17030                 elementName = "error";
17031                 break;
17032             case ResultWas::ExplicitFailure:
17033             case ResultWas::ExpressionFailed:
17034             case ResultWas::DidntThrowException:
17035                 elementName = "failure";
17036                 break;
17037
17038             // We should never see these here:
17039             case ResultWas::Info:
17040             case ResultWas::Warning:
17041             case ResultWas::Ok:
17042             case ResultWas::Unknown:
17043             case ResultWas::FailureBit:
17044             case ResultWas::Exception:
17045                 elementName = "internalError";
17046                 break;
17047         }
17048
17049         XmlWriter::ScopedElement e = xml.scopedElement( elementName );
17050
17051         xml.writeAttribute( "message", result.getExpression() );
17052         xml.writeAttribute( "type", result.getTestMacroName() );
17053
17054         ReusableStringStream rss;
17055         if (stats.totals.assertions.total() > 0) {
17056             rss << "FAILED" << ":\n";
17057             if (result.hasExpression()) {
17058                 rss << " ";
17059                 rss << result.getExpressionInMacro();
17060                 rss << '\n';
17061             }
17062             if (result.hasExpandedExpression()) {
17063                 rss << "with expansion:\n";
17064                 rss << Column(result.getExpandedExpression()).indent(2) << '\n';
17065             }
17066         } else {
17067             rss << '\n';
17068         }
17069
17070         if( !result.getMessage().empty() )
17071             rss << result.getMessage() << '\n';
17072         for( auto const& msg : stats.infoMessages )
17073             if( msg.type == ResultWas::Info )
17074                 rss << msg.message << '\n';
17075     }

```

```

17076         rss << "at " << result.getSourceInfo();
17077         xml.writeText( rss.str(), XmlFormatting::Newline );
17078     }
17079 }
17080
17081 CATCH_REGISTER_REPORTER( "junit", JunitReporter )
17082
17083 } // end namespace Catch
17084 // end catch_reporter_junit.cpp
17085 // start catch_reporter_listening.cpp
17086
17087 #include <cassert>
17088
17089 namespace Catch {
17090
17091     ListeningReporter::ListeningReporter() {
17092         // We will assume that listeners will always want all assertions
17093         m_preferences.shouldReportAllAssertions = true;
17094     }
17095
17096     void ListeningReporter::addListener( IStreamingReporterPtr&& listener ) {
17097         m_listeners.push_back( std::move( listener ) );
17098     }
17099
17100     void ListeningReporter::addReporter( IStreamingReporterPtr&& reporter ) {
17101         assert( !m_reporter && "Listening reporter can wrap only 1 real reporter" );
17102         m_reporter = std::move( reporter );
17103         m_preferences.shouldRedirectStdOut = m_reporter->getPreferences().shouldRedirectStdOut;
17104     }
17105
17106     ReporterPreferences ListeningReporter::getPreferences() const {
17107         return m_preferences;
17108     }
17109
17110     std::set<Verbosity> ListeningReporter::getSupportedVerbsities() {
17111         return std::set<Verbosity>{ };
17112     }
17113
17114     void ListeningReporter::noMatchingTestCases( std::string const& spec ) {
17115         for ( auto const& listener : m_listeners ) {
17116             listener->noMatchingTestCases( spec );
17117         }
17118         m_reporter->noMatchingTestCases( spec );
17119     }
17120
17121     void ListeningReporter::reportInvalidArguments( std::string const& arg ) {
17122         for ( auto const& listener : m_listeners ) {
17123             listener->reportInvalidArguments( arg );
17124         }
17125         m_reporter->reportInvalidArguments( arg );
17126     }
17127
17128     #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
17129     void ListeningReporter::benchmarkPreparing( std::string const& name ) {
17130         for ( auto const& listener : m_listeners ) {
17131             listener->benchmarkPreparing( name );
17132         }
17133         m_reporter->benchmarkPreparing( name );
17134     }
17135     void ListeningReporter::benchmarkStarting( BenchmarkInfo const& benchmarkInfo ) {
17136         for ( auto const& listener : m_listeners ) {
17137             listener->benchmarkStarting( benchmarkInfo );
17138         }
17139         m_reporter->benchmarkStarting( benchmarkInfo );
17140     }
17141     void ListeningReporter::benchmarkEnded( BenchmarkStats<> const& benchmarkStats ) {
17142         for ( auto const& listener : m_listeners ) {
17143             listener->benchmarkEnded( benchmarkStats );
17144         }
17145         m_reporter->benchmarkEnded( benchmarkStats );
17146     }
17147
17148     void ListeningReporter::benchmarkFailed( std::string const& error ) {
17149         for ( auto const& listener : m_listeners ) {
17150             listener->benchmarkFailed( error );
17151         }
17152         m_reporter->benchmarkFailed( error );
17153     }
17154     #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
17155
17156     void ListeningReporter::testRunStarting( TestRunInfo const& testRunInfo ) {
17157         for ( auto const& listener : m_listeners ) {
17158             listener->testRunStarting( testRunInfo );
17159         }
17160         m_reporter->testRunStarting( testRunInfo );
17161     }
17162

```

```

17163     void ListeningReporter::testGroupStarting( GroupInfo const& groupInfo ) {
17164         for ( auto const& listener : m_listeners ) {
17165             listener->testGroupStarting( groupInfo );
17166         }
17167         m_reporter->testGroupStarting( groupInfo );
17168     }
17169
17170     void ListeningReporter::testCaseStarting( TestCaseInfo const& testInfo ) {
17171         for ( auto const& listener : m_listeners ) {
17172             listener->testCaseStarting( testInfo );
17173         }
17174         m_reporter->testCaseStarting( testInfo );
17175     }
17176
17177     void ListeningReporter::sectionStarting( SectionInfo const& sectionInfo ) {
17178         for ( auto const& listener : m_listeners ) {
17179             listener->sectionStarting( sectionInfo );
17180         }
17181         m_reporter->sectionStarting( sectionInfo );
17182     }
17183
17184     void ListeningReporter::assertionStarting( AssertionInfo const& assertionInfo ) {
17185         for ( auto const& listener : m_listeners ) {
17186             listener->assertionStarting( assertionInfo );
17187         }
17188         m_reporter->assertionStarting( assertionInfo );
17189     }
17190
17191     // The return value indicates if the messages buffer should be cleared:
17192     bool ListeningReporter::assertionEnded( AssertionStats const& assertionStats ) {
17193         for( auto const& listener : m_listeners ) {
17194             static_cast<void>( listener->assertionEnded( assertionStats ) );
17195         }
17196         return m_reporter->assertionEnded( assertionStats );
17197     }
17198
17199     void ListeningReporter::sectionEnded( SectionStats const& sectionStats ) {
17200         for ( auto const& listener : m_listeners ) {
17201             listener->sectionEnded( sectionStats );
17202         }
17203         m_reporter->sectionEnded( sectionStats );
17204     }
17205
17206     void ListeningReporter::testCaseEnded( TestCaseStats const& testCaseStats ) {
17207         for ( auto const& listener : m_listeners ) {
17208             listener->testCaseEnded( testCaseStats );
17209         }
17210         m_reporter->testCaseEnded( testCaseStats );
17211     }
17212
17213     void ListeningReporter::testGroupEnded( TestGroupStats const& testGroupStats ) {
17214         for ( auto const& listener : m_listeners ) {
17215             listener->testGroupEnded( testGroupStats );
17216         }
17217         m_reporter->testGroupEnded( testGroupStats );
17218     }
17219
17220     void ListeningReporter::testRunEnded( TestRunStats const& testRunStats ) {
17221         for ( auto const& listener : m_listeners ) {
17222             listener->testRunEnded( testRunStats );
17223         }
17224         m_reporter->testRunEnded( testRunStats );
17225     }
17226
17227     void ListeningReporter::skipTest( TestCaseInfo const& testInfo ) {
17228         for ( auto const& listener : m_listeners ) {
17229             listener->skipTest( testInfo );
17230         }
17231         m_reporter->skipTest( testInfo );
17232     }
17233
17234     bool ListeningReporter::isMulti() const {
17235         return true;
17236     }
17237 } // end namespace Catch
17238 // end catch_reporter_listening.cpp
17239 // start catch_reporter_xml.cpp
17240
17241 #if defined(_MSC_VER)
17242 #pragma warning(push)
17243 #pragma warning(disable:4061) // Not all labels are EXPLICITLY handled in switch
17244 // Note that 4062 (not all labels are handled
17245 // and default is missing) is enabled
17246 #endif
17247 namespace Catch {

```

```

17250 XmlReporter::XmlReporter( ReporterConfig const& _config )
17251 :   StreamingReporterBase( _config ),
17252   m_xml(_config.stream())
17253 {
17254     m_reporterPrefs.shouldRedirectStdOut = true;
17255     m_reporterPrefs.shouldReportAllAssertions = true;
17256 }
17257
17258 XmlReporter::~XmlReporter() = default;
17259
17260 std::string XmlReporter::getDescription() {
17261     return "Reports test results as an XML document";
17262 }
17263
17264 std::string XmlReporter::getStylesheetRef() const {
17265     return std::string();
17266 }
17267
17268 void XmlReporter::writeSourceInfo( SourceLineInfo const& sourceInfo ) {
17269     m_xml
17270         .writeAttribute( "filename", sourceInfo.file )
17271         .writeAttribute( "line", sourceInfo.line );
17272 }
17273
17274 void XmlReporter::noMatchingTestCases( std::string const& s ) {
17275     StreamingReporterBase::noMatchingTestCases( s );
17276 }
17277
17278 void XmlReporter::testRunStarting( TestRunInfo const& testInfo ) {
17279     StreamingReporterBase::testRunStarting( testInfo );
17280     std::string stylesheetRef = getStylesheetRef();
17281     if( !stylesheetRef.empty() )
17282         m_xml.writeStylesheetRef( stylesheetRef );
17283     m_xml.startElement( "Catch" );
17284     if( !m_config->name().empty() )
17285         m_xml.writeAttribute( "name", m_config->name() );
17286     if( m_config->testSpec().hasFilters() )
17287         m_xml.writeAttribute( "filters", serializeFilters( m_config->getTestsOrTags() ) );
17288     if( m_config->rngSeed() != 0 )
17289         m_xml.scopedElement( "Randomness" )
17290             .writeAttribute( "seed", m_config->rngSeed() );
17291 }
17292
17293 void XmlReporter::testGroupStarting( GroupInfo const& groupInfo ) {
17294     StreamingReporterBase::testGroupStarting( groupInfo );
17295     m_xml.startElement( "Group" )
17296         .writeAttribute( "name", groupInfo.name );
17297 }
17298
17299 void XmlReporter::testCaseStarting( TestCaseInfo const& testInfo ) {
17300     StreamingReporterBase::testCaseStarting( testInfo );
17301     m_xml.startElement( "TestCase" )
17302         .writeAttribute( "name", trim( testInfo.name ) )
17303         .writeAttribute( "description", testInfo.description )
17304         .writeAttribute( "tags", testInfo.tagsAsString() );
17305
17306     writeSourceInfo( testInfo.lineInfo );
17307
17308     if ( m_config->showDurations() == ShowDurations::Always )
17309         m_testCaseTimer.start();
17310     m_xml.ensureTagClosed();
17311 }
17312
17313 void XmlReporter::sectionStarting( SectionInfo const& sectionInfo ) {
17314     StreamingReporterBase::sectionStarting( sectionInfo );
17315     if( m_sectionDepth++ > 0 ) {
17316         m_xml.startElement( "Section" )
17317             .writeAttribute( "name", trim( sectionInfo.name ) );
17318         writeSourceInfo( sectionInfo.lineInfo );
17319         m_xml.ensureTagClosed();
17320     }
17321 }
17322
17323 void XmlReporter::assertionStarting( AssertionInfo const& ) { }
17324
17325 bool XmlReporter::assertionEnded( AssertionStats const& assertionStats ) {
17326
17327     AssertionResult const& result = assertionStats.assertionResult;
17328
17329     bool includeResults = m_config->includeSuccessfulResults() || !result.isOk();
17330
17331     if( includeResults || result.getResultType() == ResultWas::Warning ) {
17332         // Print any info messages in <Info> tags.
17333         for( auto const& msg : assertionStats.infoMessages ) {
17334             if( msg.type == ResultWas::Info && includeResults ) {
17335                 m_xml.scopedElement( "Info" )
17336                     .writeText( msg.message );

```

```

17337         } else if ( msg.type == ResultWas::Warning ) {
17338             m_xml.scopedElement( "Warning" )
17339                 .writeText( msg.message );
17340         }
17341     }
17342 }
17343
17344 // Drop out if result was successful but we're not printing them.
17345 if( !includeResults && result.getResultType() != ResultWas::Warning )
17346     return true;
17347
17348 // Print the expression if there is one.
17349 if( result.hasExpression() ) {
17350     m_xml.startElement( "Expression" )
17351         .writeAttribute( "success", result.succeeded() )
17352         .writeAttribute( "type", result.getTestMacroName() );
17353
17354     writeSourceInfo( result.getSourceInfo() );
17355
17356     m_xml.scopedElement( "Original" )
17357         .writeText( result.getExpression() );
17358     m_xml.scopedElement( "Expanded" )
17359         .writeText( result.getExpandedExpression() );
17360 }
17361
17362 // And... Print a result applicable to each result type.
17363 switch( result.getResultType() ) {
17364     case ResultWas::ThrowException:
17365         m_xml.startElement( "Exception" );
17366         writeSourceInfo( result.getSourceInfo() );
17367         m_xml.writeText( result.getMessage() );
17368         m_xml.endElement();
17369         break;
17370     case ResultWas::FatalErrorCondition:
17371         m_xml.startElement( "FatalErrorCondition" );
17372         writeSourceInfo( result.getSourceInfo() );
17373         m_xml.writeText( result.getMessage() );
17374         m_xml.endElement();
17375         break;
17376     case ResultWas::Info:
17377         m_xml.scopedElement( "Info" )
17378             .writeText( result.getMessage() );
17379         break;
17380     case ResultWas::Warning:
17381         // Warning will already have been written
17382         break;
17383     case ResultWas::ExplicitFailure:
17384         m_xml.startElement( "Failure" );
17385         writeSourceInfo( result.getSourceInfo() );
17386         m_xml.writeText( result.getMessage() );
17387         m_xml.endElement();
17388         break;
17389     default:
17390         break;
17391 }
17392
17393 if( result.hasExpression() )
17394     m_xml.endElement();
17395
17396 return true;
17397 }
17398
17399 void XmlReporter::sectionEnded( SectionStats const& sectionStats ) {
17400     StreamingReporterBase::sectionEnded( sectionStats );
17401     if( --m_sectionDepth > 0 ) {
17402         XmlWriter::ScopedElement e = m_xml.scopedElement( "OverallResults" );
17403         e.writeAttribute( "successes", sectionStats.assertions.passed );
17404         e.writeAttribute( "failures", sectionStats.assertions.failed );
17405         e.writeAttribute( "expectedFailures", sectionStats.assertions.failedButOk );
17406
17407         if ( m_config->showDurations() == ShowDurations::Always )
17408             e.writeAttribute( "durationInSeconds", sectionStats.durationInSeconds );
17409
17410         m_xml.endElement();
17411     }
17412 }
17413
17414 void XmlReporter::testCaseEnded( TestCaseStats const& testCaseStats ) {
17415     StreamingReporterBase::testCaseEnded( testCaseStats );
17416     XmlWriter::ScopedElement e = m_xml.scopedElement( "OverallResult" );
17417     e.writeAttribute( "success", testCaseStats.totals.assertions.allOk() );
17418
17419     if ( m_config->showDurations() == ShowDurations::Always )
17420         e.writeAttribute( "durationInSeconds", m_testCaseTimer.getElapsedSeconds() );
17421
17422     if( !testCaseStats.stdOut.empty() )
17423         m_xml.scopedElement( "StdOut" ).writeText( trim( testCaseStats.stdOut ),

```

```

        XmlFormatting::Newline );
17424         if ( !testCaseStats.stdErr.empty() )
17425             m_xml.scopedElement( "StdErr" ).writeText( trim( testCaseStats.stdErr ),
        XmlFormatting::Newline );
17426
17427         m_xml.endElement();
17428     }
17429
17430     void XmlReporter::testGroupEnded( TestGroupStats const& testGroupStats ) {
17431         StreamingReporterBase::testGroupEnded( testGroupStats );
17432         // TODO: Check testGroupStats.aborting and act accordingly.
17433         m_xml.scopedElement( "OverallResults" )
17434             .writeAttribute( "successes", testGroupStats.totals.assertions.passed )
17435             .writeAttribute( "failures", testGroupStats.totals.assertions.failed )
17436             .writeAttribute( "expectedFailures", testGroupStats.totals.assertions.failedButOk );
17437         m_xml.scopedElement( "OverallResultsCases" )
17438             .writeAttribute( "successes", testGroupStats.totals.testCases.passed )
17439             .writeAttribute( "failures", testGroupStats.totals.testCases.failed )
17440             .writeAttribute( "expectedFailures", testGroupStats.totals.testCases.failedButOk );
17441         m_xml.endElement();
17442     }
17443
17444     void XmlReporter::testRunEnded( TestRunStats const& testRunStats ) {
17445         StreamingReporterBase::testRunEnded( testRunStats );
17446         m_xml.scopedElement( "OverallResults" )
17447             .writeAttribute( "successes", testRunStats.totals.assertions.passed )
17448             .writeAttribute( "failures", testRunStats.totals.assertions.failed )
17449             .writeAttribute( "expectedFailures", testRunStats.totals.assertions.failedButOk );
17450         m_xml.scopedElement( "OverallResultsCases" )
17451             .writeAttribute( "successes", testRunStats.totals.testCases.passed )
17452             .writeAttribute( "failures", testRunStats.totals.testCases.failed )
17453             .writeAttribute( "expectedFailures", testRunStats.totals.testCases.failedButOk );
17454         m_xml.endElement();
17455     }
17456
17457     #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
17458     void XmlReporter::benchmarkPreparing( std::string const& name ) {
17459         m_xml.startElement( "BenchmarkResults" )
17460             .writeAttribute( "name", name );
17461     }
17462
17463     void XmlReporter::benchmarkStarting( BenchmarkInfo const& info ) {
17464         m_xml.writeAttribute( "samples", info.samples )
17465             .writeAttribute( "resamples", info.resamples )
17466             .writeAttribute( "iterations", info.iterations )
17467             .writeAttribute( "clockResolution", info.clockResolution )
17468             .writeAttribute( "estimatedDuration", info.estimatedDuration )
17469             .writeComment( "All values in nano seconds" );
17470     }
17471
17472     void XmlReporter::benchmarkEnded( BenchmarkStats<> const& benchmarkStats ) {
17473         m_xml.startElement( "mean" )
17474             .writeAttribute( "value", benchmarkStats.mean.point.count() )
17475             .writeAttribute( "lowerBound", benchmarkStats.mean.lower_bound.count() )
17476             .writeAttribute( "upperBound", benchmarkStats.mean.upper_bound.count() )
17477             .writeAttribute( "ci", benchmarkStats.mean.confidence_interval );
17478         m_xml.endElement();
17479         m_xml.startElement( "standardDeviation" )
17480             .writeAttribute( "value", benchmarkStats.standardDeviation.point.count() )
17481             .writeAttribute( "lowerBound", benchmarkStats.standardDeviation.lower_bound.count() )
17482             .writeAttribute( "upperBound", benchmarkStats.standardDeviation.upper_bound.count() )
17483             .writeAttribute( "ci", benchmarkStats.standardDeviation.confidence_interval );
17484         m_xml.endElement();
17485         m_xml.startElement( "outliers" )
17486             .writeAttribute( "variance", benchmarkStats.outlierVariance )
17487             .writeAttribute( "lowMild", benchmarkStats.outliers.low_mild )
17488             .writeAttribute( "lowSevere", benchmarkStats.outliers.low_severe )
17489             .writeAttribute( "highMild", benchmarkStats.outliers.high_mild )
17490             .writeAttribute( "highSevere", benchmarkStats.outliers.high_severe );
17491         m_xml.endElement();
17492     }
17493 }
17494
17495 void XmlReporter::benchmarkFailed( std::string const& error ) {
17496     m_xml.scopedElement( "failed" )
17497         .writeAttribute( "message", error );
17498     m_xml.endElement();
17499 }
17500 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
17501
17502     CATCH_REGISTER_REPORTER( "xml", XmlReporter )
17503
17504 } // end namespace Catch
17505
17506 #if defined(_MSC_VER)
17507 #pragma warning(pop)
17508 #endif

```

```

17509 // end catch_reporter_xml.cpp
17510
17511 namespace Catch {
17512     LeakDetector leakDetector;
17513 }
17514
17515 #ifdef __clang__
17516 #pragma clang diagnostic pop
17517 #endif
17518
17519 // end catch_impl.hpp
17520 #endif
17521
17522 #ifdef CATCH_CONFIG_MAIN
17523 // start catch_default_main.hpp
17524
17525 #ifndef __OBJC__
17526
17527 #ifndef CATCH_INTERNAL_CDECL
17528 #ifdef _MSC_VER
17529 #define CATCH_INTERNAL_CDECL __cdecl
17530 #else
17531 #define CATCH_INTERNAL_CDECL
17532 #endif
17533 #endif
17534
17535 #if defined(CATCH_CONFIG_WCHAR) && defined(CATCH_PLATFORM_WINDOWS) && defined(_UNICODE) &&
!defined(DO_NOT_USE_WMAIN)
17536 // Standard C/C++ Win32 Unicode wmain entry point
17537 extern "C" int CATCH_INTERNAL_CDECL wmain (int argc, wchar_t * argv[], wchar_t * []) {
17538 #else
17539 // Standard C/C++ main entry point
17540 int CATCH_INTERNAL_CDECL main (int argc, char * argv[]) {
17541 #endif
17542
17543     return Catch::Session().run( argc, argv );
17544 }
17545
17546 #else // __OBJC__
17547 // Objective-C entry point
17548 int main (int argc, char * const argv[]) {
17549 #if !CATCH_ARC_ENABLED
17550     NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
17551 #endif
17552
17553     Catch::registerTestMethods();
17554     int result = Catch::Session().run( argc, (char**)argv );
17555
17556 #if !CATCH_ARC_ENABLED
17557     [pool drain];
17558 #endif
17559
17560     return result;
17561 }
17562 #endif // __OBJC__
17563
17564 // end catch_default_main.hpp
17565 #endif
17566
17567 #if !defined(CATCH_CONFIG_IMPL_ONLY)
17568
17569 #ifndef CLARA_CONFIG_MAIN_NOT_DEFINED
17570 #undef CLARA_CONFIG_MAIN
17571 #endif
17572
17573 #if !defined(CATCH_CONFIG_DISABLE)
17574 // If this config identifier is defined then all CATCH macros are prefixed with CATCH_
17575 #ifdef CATCH_CONFIG_PREFIX_ALL
17576 #define CATCH_REQUIRE( ... ) INTERNAL_CATCH_TEST( "CATCH_REQUIRE", Catch::ResultDisposition::Normal,
__VA_ARGS__ )
17577 #define CATCH_REQUIRE_FALSE( ... ) INTERNAL_CATCH_TEST( "CATCH_REQUIRE_FALSE",
Catch::ResultDisposition::Normal | Catch::ResultDisposition::FalseTest, __VA_ARGS__ )
17578 #define CATCH_REQUIRE_THROWS( ... ) INTERNAL_CATCH_THROWS( "CATCH_REQUIRE_THROWS",
Catch::ResultDisposition::Normal, __VA_ARGS__ )
17579 #define CATCH_REQUIRE_THROWS_AS( expr, exceptionType ) INTERNAL_CATCH_THROWS_AS(
"CATCH_REQUIRE_THROWS_AS", exceptionType, Catch::ResultDisposition::Normal, expr )
17580 #define CATCH_REQUIRE_THROWS_WITH( expr, matcher ) INTERNAL_CATCH_THROWS_STR_MATCHES(
"CATCH_REQUIRE_THROWS_WITH", Catch::ResultDisposition::Normal, matcher, expr )
17581 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17582 #define CATCH_REQUIRE_THROWS_MATCHES( expr, exceptionType, matcher ) INTERNAL_CATCH_THROWS_MATCHES(
"CATCH_REQUIRE_THROWS_MATCHES", exceptionType, Catch::ResultDisposition::Normal, matcher, expr )
17583 #endif
17584 #define CATCH_REQUIRE_NO_THROW( ... ) INTERNAL_CATCH_NO_THROW( "CATCH_REQUIRE_NO_THROW",

```

```

    Catch::ResultDisposition::Normal, __VA_ARGS__ )
17590
17591 #define CATCH_CHECK( ... ) INTERNAL_CATCH_TEST( "CATCH_CHECK",
    Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17592 #define CATCH_CHECK_FALSE( ... ) INTERNAL_CATCH_TEST( "CATCH_CHECK_FALSE",
    Catch::ResultDisposition::ContinueOnFailure | Catch::ResultDisposition::FalseTest, __VA_ARGS__ )
17593 #define CATCH_CHECKED_IF( ... ) INTERNAL_CATCH_IF( "CATCH_CHECKED_IF",
    Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17594 #define CATCH_CHECKED_ELSE( ... ) INTERNAL_CATCH_ELSE( "CATCH_CHECKED_ELSE",
    Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17595 #define CATCH_CHECK_NOFAIL( ... ) INTERNAL_CATCH_TEST( "CATCH_CHECK_NOFAIL",
    Catch::ResultDisposition::ContinueOnFailure | Catch::ResultDisposition::SuppressFail, __VA_ARGS__ )
17596
17597 #define CATCH_CHECK_THROWS( ... ) INTERNAL_CATCH_THROWS( "CATCH_CHECK_THROWS",
    Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17598 #define CATCH_CHECK_THROWS_AS( expr, exceptionType ) INTERNAL_CATCH_THROWS_AS(
    "CATCH_CHECK_THROWS_AS", exceptionType, Catch::ResultDisposition::ContinueOnFailure, expr )
17599 #define CATCH_CHECK_THROWS_WITH( expr, matcher ) INTERNAL_CATCH_THROWS_STR_MATCHES(
    "CATCH_CHECK_THROWS_WITH", Catch::ResultDisposition::ContinueOnFailure, matcher, expr )
17600 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17601 #define CATCH_CHECK_THROWS_MATCHES( expr, exceptionType, matcher ) INTERNAL_CATCH_THROWS_MATCHES(
    "CATCH_CHECK_THROWS_MATCHES", exceptionType, Catch::ResultDisposition::ContinueOnFailure, matcher,
    expr )
17602 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17603 #define CATCH_CHECK_NOTHROW( ... ) INTERNAL_CATCH_NO_THROW( "CATCH_CHECK_NOTHROW",
    Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17604
17605 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17606 #define CATCH_CHECK_THAT( arg, matcher ) INTERNAL_CHECK_THAT( "CATCH_CHECK_THAT", matcher,
    Catch::ResultDisposition::ContinueOnFailure, arg )
17607
17608 #define CATCH_REQUIRE_THAT( arg, matcher ) INTERNAL_CHECK_THAT( "CATCH_REQUIRE_THAT", matcher,
    Catch::ResultDisposition::Normal, arg )
17609 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17610
17611 #define CATCH_INFO( msg ) INTERNAL_CATCH_INFO( "CATCH_INFO", msg )
17612 #define CATCH_UNSCOPED_INFO( msg ) INTERNAL_CATCH_UNSCOPED_INFO( "CATCH_UNSCOPED_INFO", msg )
17613 #define CATCH_WARN( msg ) INTERNAL_CATCH_MSG( "CATCH_WARN", Catch::ResultWas::Warning,
    Catch::ResultDisposition::ContinueOnFailure, msg )
17614 #define CATCH_CAPTURE( ... ) INTERNAL_CATCH_CAPTURE( INTERNAL_CATCH_UNIQUE_NAME(capturer),
    "CATCH_CAPTURE", __VA_ARGS__ )
17615
17616 #define CATCH_TEST_CASE( ... ) INTERNAL_CATCH_TESTCASE( __VA_ARGS__ )
17617 #define CATCH_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEST_CASE_METHOD( className,
    __VA_ARGS__ )
17618 #define CATCH_METHOD_AS_TEST_CASE( method, ... ) INTERNAL_CATCH_METHOD_AS_TEST_CASE( method,
    __VA_ARGS__ )
17619 #define CATCH_REGISTER_TEST_CASE( Function, ... ) INTERNAL_CATCH_REGISTER_TESTCASE( Function,
    __VA_ARGS__ )
17620 #define CATCH_SECTION( ... ) INTERNAL_CATCH_SECTION( __VA_ARGS__ )
17621 #define CATCH_DYNAMIC_SECTION( ... ) INTERNAL_CATCH_DYNAMIC_SECTION( __VA_ARGS__ )
17622 #define CATCH_FAIL( ... ) INTERNAL_CATCH_MSG( "CATCH_FAIL", Catch::ResultWas::ExplicitFailure,
    Catch::ResultDisposition::Normal, __VA_ARGS__ )
17623 #define CATCH_FAIL_CHECK( ... ) INTERNAL_CATCH_MSG( "CATCH_FAIL_CHECK",
    Catch::ResultWas::ExplicitFailure, Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17624 #define CATCH_SUCCEED( ... ) INTERNAL_CATCH_MSG( "CATCH_SUCCEED", Catch::ResultWas::Ok,
    Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17625
17626 #define CATCH_ANON_TEST_CASE() INTERNAL_CATCH_TESTCASE()
17627
17628 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
17629 #define CATCH_TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
17630 #define CATCH_TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG( __VA_ARGS__ )
17631 #define CATCH_TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD(
    className, __VA_ARGS__ )
17632 #define CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, ... )
    INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ )
17633 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE( __VA_ARGS__ )
17634 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG(
    __VA_ARGS__ )
17635 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... )
    INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, __VA_ARGS__ )
17636 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... )
    INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ )
17637 #else
17638 #define CATCH_TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS(
    INTERNAL_CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ ) )
17639 #define CATCH_TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS(
    INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG( __VA_ARGS__ ) )
17640 #define CATCH_TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
    INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD( className, __VA_ARGS__ ) )
17641 #define CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
    INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ ) )
17642 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS(
    INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE( __VA_ARGS__ ) )
17643 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS(

```



```

INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( __VA_ARGS__ )
17644 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, __VA_ARGS__ ) )
17645 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ ) )
17646 #endif
17647
17648 #if !defined(CATCH_CONFIG_RUNTIME_STATIC_REQUIRE)
17649 #define CATCH_STATIC_REQUIRE( ... )          static_assert( __VA_ARGS__ , #__VA_ARGS__ );
CATCH_SUCCEED( #__VA_ARGS__ )
17650 #define CATCH_STATIC_REQUIRE_FALSE( ... ) static_assert( !( __VA_ARGS__ ), "!( " #__VA_ARGS__ " )" );
CATCH_SUCCEED( #__VA_ARGS__ )
17651 #else
17652 #define CATCH_STATIC_REQUIRE( ... )          CATCH_REQUIRE( __VA_ARGS__ )
17653 #define CATCH_STATIC_REQUIRE_FALSE( ... )    CATCH_REQUIRE_FALSE( __VA_ARGS__ )
17654 #endif
17655
17656 // "BDD-style" convenience wrappers
17657 #define CATCH_SCENARIO( ... ) CATCH_TEST_CASE( "Scenario: " __VA_ARGS__ )
17658 #define CATCH_SCENARIO_METHOD( className, ... ) INTERNAL_CATCH_TEST_CASE_METHOD( className, "Scenario:
" __VA_ARGS__ )
17659 #define CATCH_GIVEN( desc )          INTERNAL_CATCH_DYNAMIC_SECTION( "    Given: " « desc )
17660 #define CATCH_AND_GIVEN( desc )      INTERNAL_CATCH_DYNAMIC_SECTION( "And given: " « desc )
17661 #define CATCH_WHEN( desc )           INTERNAL_CATCH_DYNAMIC_SECTION( "    When: " « desc )
17662 #define CATCH_AND_WHEN( desc )       INTERNAL_CATCH_DYNAMIC_SECTION( " And when: " « desc )
17663 #define CATCH_THEN( desc )           INTERNAL_CATCH_DYNAMIC_SECTION( "    Then: " « desc )
17664 #define CATCH_AND_THEN( desc )       INTERNAL_CATCH_DYNAMIC_SECTION( " And: " « desc )
17665
17666 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
17667 #define CATCH_BENCHMARK(...) \
17668     INTERNAL_CATCH_BENCHMARK(INTERNAL_CATCH_UNIQUE_NAME(C_A_T_C_H_B_E_N_C_H_),
INTERNAL_CATCH_GET_1_ARG(__VA_ARGS__), INTERNAL_CATCH_GET_2_ARG(__VA_ARGS__))
17669 #define CATCH_BENCHMARK_ADVANCED( name ) \
17670     INTERNAL_CATCH_BENCHMARK_ADVANCED(INTERNAL_CATCH_UNIQUE_NAME(C_A_T_C_H_B_E_N_C_H_), name)
17671 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
17672
17673 // If CATCH_CONFIG_PREFIX_ALL is not defined then the CATCH_ prefix is not required
17674 #else
17675
17676 #define REQUIRE( ... ) INTERNAL_CATCH_TEST( "REQUIRE", Catch::ResultDisposition::Normal, __VA_ARGS__
)
17677 #define REQUIRE_FALSE( ... ) INTERNAL_CATCH_TEST( "REQUIRE_FALSE", Catch::ResultDisposition::Normal |
Catch::ResultDisposition::FalseTest, __VA_ARGS__ )
17678
17679 #define REQUIRE_THROWS( ... ) INTERNAL_CATCH_THROWS( "REQUIRE_THROWS",
Catch::ResultDisposition::Normal, __VA_ARGS__ )
17680 #define REQUIRE_THROWS_AS( expr, exceptionType ) INTERNAL_CATCH_THROWS_AS( "REQUIRE_THROWS_AS",
exceptionType, Catch::ResultDisposition::Normal, expr )
17681 #define REQUIRE_THROWS_WITH( expr, matcher ) INTERNAL_CATCH_THROWS_STR_MATCHES( "REQUIRE_THROWS_WITH",
Catch::ResultDisposition::Normal, matcher, expr )
17682 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17683 #define REQUIRE_THROWS_MATCHES( expr, exceptionType, matcher ) INTERNAL_CATCH_THROWS_MATCHES(
"REQUIRE_THROWS_MATCHES", exceptionType, Catch::ResultDisposition::Normal, matcher, expr )
17684 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17685 #define REQUIRE_NO_THROW( ... ) INTERNAL_CATCH_NO_THROW( "REQUIRE_NO_THROW",
Catch::ResultDisposition::Normal, __VA_ARGS__ )
17686
17687 #define CHECK( ... ) INTERNAL_CATCH_TEST( "CHECK", Catch::ResultDisposition::ContinueOnFailure,
__VA_ARGS__ )
17688 #define CHECK_FALSE( ... ) INTERNAL_CATCH_TEST( "CHECK_FALSE",
Catch::ResultDisposition::ContinueOnFailure | Catch::ResultDisposition::FalseTest, __VA_ARGS__ )
17689 #define CHECKED_IF( ... ) INTERNAL_CATCH_IF( "CHECKED_IF",
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17690 #define CHECKED_ELSE( ... ) INTERNAL_CATCH_ELSE( "CHECKED_ELSE",
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17691 #define CHECK_NO_FAIL( ... ) INTERNAL_CATCH_TEST( "CHECK_NO_FAIL",
Catch::ResultDisposition::ContinueOnFailure | Catch::ResultDisposition::SuppressFail, __VA_ARGS__ )
17692
17693 #define CHECK_THROWS( ... ) INTERNAL_CATCH_THROWS( "CHECK_THROWS",
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17694 #define CHECK_THROWS_AS( expr, exceptionType ) INTERNAL_CATCH_THROWS_AS( "CHECK_THROWS_AS",
exceptionType, Catch::ResultDisposition::ContinueOnFailure, expr )
17695 #define CHECK_THROWS_WITH( expr, matcher ) INTERNAL_CATCH_THROWS_STR_MATCHES( "CHECK_THROWS_WITH",
Catch::ResultDisposition::ContinueOnFailure, matcher, expr )
17696 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17697 #define CHECK_THROWS_MATCHES( expr, exceptionType, matcher ) INTERNAL_CATCH_THROWS_MATCHES(
"CHECK_THROWS_MATCHES", exceptionType, Catch::ResultDisposition::ContinueOnFailure, matcher, expr )
17698 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17699 #define CHECK_NO_THROW( ... ) INTERNAL_CATCH_NO_THROW( "CHECK_NO_THROW",
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17700
17701 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17702 #define CHECK_THAT( arg, matcher ) INTERNAL_CHECK_THAT( "CHECK_THAT", matcher,
Catch::ResultDisposition::ContinueOnFailure, arg )
17703
17704 #define REQUIRE_THAT( arg, matcher ) INTERNAL_CHECK_THAT( "REQUIRE_THAT", matcher,
Catch::ResultDisposition::Normal, arg )

```

```

17705 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17706
17707 #define INFO( msg ) INTERNAL_CATCH_INFO( "INFO", msg )
17708 #define UNSCOPED_INFO( msg ) INTERNAL_CATCH_UNSCOPED_INFO( "UNSCOPED_INFO", msg )
17709 #define WARN( msg ) INTERNAL_CATCH_MSG( "WARN", Catch::ResultWas::Warning,
17710 Catch::ResultDisposition::ContinueOnFailure, msg )
17711 #define CAPTURE( ... ) INTERNAL_CATCH_CAPTURE( INTERNAL_CATCH_UNIQUE_NAME(capturer),
17712 "CAPTURE",__VA_ARGS__ )
17713
17714 #define TEST_CASE( ... ) INTERNAL_CATCH_TESTCASE( __VA_ARGS__ )
17715 #define TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEST_CASE_METHOD( className, __VA_ARGS__ )
17716 #define METHOD_AS_TEST_CASE( method, ... ) INTERNAL_CATCH_METHOD_AS_TEST_CASE( method, __VA_ARGS__ )
17717 #define REGISTER_TEST_CASE( Function, ... ) INTERNAL_CATCH_REGISTER_TESTCASE( Function, __VA_ARGS__ )
17718 #define SECTION( ... ) INTERNAL_CATCH_SECTION( __VA_ARGS__ )
17719 #define DYNAMIC_SECTION( ... ) INTERNAL_CATCH_DYNAMIC_SECTION( __VA_ARGS__ )
17720 #define FAIL( ... ) INTERNAL_CATCH_MSG( "FAIL", Catch::ResultWas::ExplicitFailure,
17721 Catch::ResultDisposition::Normal, __VA_ARGS__ )
17722 #define FAIL_CHECK( ... ) INTERNAL_CATCH_MSG( "FAIL_CHECK", Catch::ResultWas::ExplicitFailure,
17723 Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17724 #define SUCCEED( ... ) INTERNAL_CATCH_MSG( "SUCCEED", Catch::ResultWas::Ok,
17725 Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17726 #define ANON_TEST_CASE() INTERNAL_CATCH_TESTCASE()
17727
17728 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
17729 #define TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
17730 #define TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG( __VA_ARGS__ )
17731 #define TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD(
17732 className, __VA_ARGS__ )
17733 #define TEMPLATE_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG(
17734 className, __VA_ARGS__ )
17735 #define TEMPLATE_PRODUCT_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE( __VA_ARGS__ )
17736 #define TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG(
17737 __VA_ARGS__ )
17738 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... )
17739 INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, __VA_ARGS__ )
17740 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... )
17741 INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ )
17742 #define TEMPLATE_LIST_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE( __VA_ARGS__ )
17743 #define TEMPLATE_LIST_TEST_CASE_METHOD( className, ... )
17744 INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD( className, __VA_ARGS__ )
17745 #define TEMPLATE_LIST_TEST_CASE_METHOD_SIG( className, ... )
17746 INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ )
17747 #define TEMPLATE_LIST_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_SIG(
17748 __VA_ARGS__ )
17749 #define TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG(
17750 __VA_ARGS__ )
17751 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... )
17752 INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ )
17753 #define TEMPLATE_LIST_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_SIG(
17754 __VA_ARGS__ )
17755 #define TEMPLATE_LIST_TEST_CASE_METHOD_SIG( className, ... )
17756 INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ )
17757 #endif
17758
17759 #if !defined(CATCH_CONFIG_RUNTIME_STATIC_REQUIRE)
17760 #define STATIC_REQUIRE( ... ) static_assert( __VA_ARGS__, #__VA_ARGS__ ); SUCCEED(
17761 #__VA_ARGS__ )
17762 #define STATIC_REQUIRE_FALSE( ... ) static_assert( !( __VA_ARGS__ ), "!( " #__VA_ARGS__ " ) ); SUCCEED(
17763 "!( " #__VA_ARGS__ " ) )
17764 #else
17765 #define STATIC_REQUIRE( ... ) REQUIRE( __VA_ARGS__ )
17766 #define STATIC_REQUIRE_FALSE( ... ) REQUIRE_FALSE( __VA_ARGS__ )
17767 #endif
17768
17769 #endif
17770
17771 #define CATCH_TRANSLATE_EXCEPTION( signature ) INTERNAL_CATCH_TRANSLATE_EXCEPTION( signature )
17772
17773 // "BDD-style" convenience wrappers
17774 #define SCENARIO( ... ) TEST_CASE( "Scenario: " __VA_ARGS__ )
17775 #define SCENARIO_METHOD( className, ... ) INTERNAL_CATCH_TEST_CASE_METHOD( className, "Scenario: "
17776 __VA_ARGS__ )
17777
17778 #define GIVEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( " Given: " « desc )
17779 #define AND_GIVEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( "And given: " « desc )
17780 #define WHEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( " When: " « desc )
17781 #define AND_WHEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( " And when: " « desc )
17782 #define THEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( " Then: " « desc )

```

```

17768 #define AND_THEN( desc )    INTERNAL_CATCH_DYNAMIC_SECTION( "        And: " « desc )
17769
17770 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
17771 #define BENCHMARK(...) \
17772     INTERNAL_CATCH_BENCHMARK(INTERNAL_CATCH_UNIQUE_NAME(C_A_T_C_H_B_E_N_C_H_),
17773     INTERNAL_CATCH_GET_1_ARG(__VA_ARGS__,), INTERNAL_CATCH_GET_2_ARG(__VA_ARGS__,))
17774 #define BENCHMARK_ADVANCED(name) \
17775     INTERNAL_CATCH_BENCHMARK_ADVANCED(INTERNAL_CATCH_UNIQUE_NAME(C_A_T_C_H_B_E_N_C_H_), name)
17776 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
17777 using Catch::Detail::Approx;
17778
17779 #else // CATCH_CONFIG_DISABLE
17780
17781 // If this config identifier is defined then all CATCH macros are prefixed with CATCH_
17782 #ifndef CATCH_CONFIG_PREFIX_ALL
17783 #define CATCH_REQUIRE( ... )           (void)(0)
17784 #define CATCH_REQUIRE_FALSE( ... )     (void)(0)
17785 #define CATCH_REQUIRE_THROWS( ... )    (void)(0)
17786 #define CATCH_REQUIRE_THROWS_AS( expr, exceptionType ) (void)(0)
17787 #define CATCH_REQUIRE_THROWS_WITH( expr, matcher )      (void)(0)
17788 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17789 #define CATCH_REQUIRE_THROWS_MATCHES( expr, exceptionType, matcher ) (void)(0)
17790 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17791 #define CATCH_REQUIRE_NO_THROW( ... ) (void)(0)
17792 #define CATCH_CHECK( ... )             (void)(0)
17793 #define CATCH_CHECK_FALSE( ... )        (void)(0)
17794 #define CATCH_CHECKED_IF( ... )         if (__VA_ARGS__)
17795 #define CATCH_CHECKED_ELSE( ... )       if (!(__VA_ARGS__))
17796 #define CATCH_CHECK_NOFAIL( ... )       (void)(0)
17797 #define CATCH_CHECK_THROWS( ... )       (void)(0)
17798 #define CATCH_CHECK_THROWS_AS( expr, exceptionType ) (void)(0)
17799 #define CATCH_CHECK_THROWS_WITH( expr, matcher )      (void)(0)
17800 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17801 #define CATCH_CHECK_THROWS_MATCHES( expr, exceptionType, matcher ) (void)(0)
17802 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17803 #define CATCH_CHECK_NO_THROW( ... )      (void)(0)
17804 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17805 #define CATCH_CHECK_THAT( arg, matcher ) (void)(0)
17806 #endif
17807 #define CATCH_REQUIRE_THAT( arg, matcher ) (void)(0)
17808 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17809 #define CATCH_INFO( msg )                 (void)(0)
17810 #define CATCH_UNSCOPED_INFO( msg )        (void)(0)
17811 #define CATCH_WARN( msg )                 (void)(0)
17812 #define CATCH_CAPTURE( msg )              (void)(0)
17813
17814 #define CATCH_TEST_CASE( ... ) INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
17815     C_A_T_C_H_T_E_S_T_ ))
17816 #define CATCH_TEST_CASE_METHOD( className, ... )
17817     INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ))
17818 #define CATCH_METHOD_AS_TEST_CASE( method, ... ) (void)(0)
17819 #define CATCH_REGISTER_TEST_CASE( Function, ... ) (void)(0)
17820 #define CATCH_SECTION( ... )
17821 #define CATCH_DYNAMIC_SECTION( ... )
17822 #define CATCH_FAIL( ... ) (void)(0)
17823 #define CATCH_FAIL_CHECK( ... ) (void)(0)
17824 #define CATCH_SUCCEED( ... ) (void)(0)
17825
17826 #define CATCH_ANON_TEST_CASE() INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
17827     C_A_T_C_H_T_E_S_T_ ))
17828
17829 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
17830 #define CATCH_TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION(__VA_ARGS__)
17831 #define CATCH_TEMPLATE_TEST_CASE_SIG( ... )
17832     INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION(__VA_ARGS__)
17833 #define CATCH_TEMPLATE_TEST_CASE_METHOD( className, ... )
17834     INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION(className, __VA_ARGS__)
17835 #define CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, ... )
17836     INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION(className, __VA_ARGS__)
17837 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE( ... ) CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
17838 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
17839 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) CATCH_TEMPLATE_TEST_CASE_METHOD(
17840     className, __VA_ARGS__ )
17841 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) CATCH_TEMPLATE_TEST_CASE_METHOD(
17842     className, __VA_ARGS__ )
17843 #else
17844 #define CATCH_TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS(
17845     INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION(__VA_ARGS__) )
17846 #define CATCH_TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS(
17847     INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION(__VA_ARGS__) )

```

```

17845 #define CATCH_TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS (
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION(className, __VA_ARGS__ ) )
17846 #define CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS (
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION(className, __VA_ARGS__ ) )
17847 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE( ... ) CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
17848 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
17849 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) CATCH_TEMPLATE_TEST_CASE_METHOD(
className, __VA_ARGS__ )
17850 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) CATCH_TEMPLATE_TEST_CASE_METHOD(
className, __VA_ARGS__ )
17851 #endif
17852
17853 // "BDD-style" convenience wrappers
17854 #define CATCH_SCENARIO( ... ) INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_S_T_ ))
17855 #define CATCH_SCENARIO_METHOD( className, ... )
INTERNAL_CATCH_TESTCASE_METHOD_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ),
className )
17856 #define CATCH_GIVEN( desc )
17857 #define CATCH_AND_GIVEN( desc )
17858 #define CATCH_WHEN( desc )
17859 #define CATCH_AND_WHEN( desc )
17860 #define CATCH_THEN( desc )
17861 #define CATCH_AND_THEN( desc )
17862
17863 #define CATCH_STATIC_REQUIRE( ... ) (void)(0)
17864 #define CATCH_STATIC_REQUIRE_FALSE( ... ) (void)(0)
17865
17866 // If CATCH_CONFIG_PREFIX_ALL is not defined then the CATCH_ prefix is not required
17867 #else
17868
17869 #define REQUIRE( ... ) (void)(0)
17870 #define REQUIRE_FALSE( ... ) (void)(0)
17871
17872 #define REQUIRE_THROWS( ... ) (void)(0)
17873 #define REQUIRE_THROWS_AS( expr, exceptionType ) (void)(0)
17874 #define REQUIRE_THROWS_WITH( expr, matcher ) (void)(0)
17875 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17876 #define REQUIRE_THROWS_MATCHES( expr, exceptionType, matcher ) (void)(0)
17877 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17878 #define REQUIRE_NOTHROW( ... ) (void)(0)
17879
17880 #define CHECK( ... ) (void)(0)
17881 #define CHECK_FALSE( ... ) (void)(0)
17882 #define CHECKED_IF( ... ) if ( __VA_ARGS__ )
17883 #define CHECKED_ELSE( ... ) if ( !( __VA_ARGS__ ) )
17884 #define CHECK_NOFAIL( ... ) (void)(0)
17885
17886 #define CHECK_THROWS( ... ) (void)(0)
17887 #define CHECK_THROWS_AS( expr, exceptionType ) (void)(0)
17888 #define CHECK_THROWS_WITH( expr, matcher ) (void)(0)
17889 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17890 #define CHECK_THROWS_MATCHES( expr, exceptionType, matcher ) (void)(0)
17891 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17892 #define CHECK_NOTHROW( ... ) (void)(0)
17893
17894 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17895 #define CHECK_THAT( arg, matcher ) (void)(0)
17896
17897 #define REQUIRE_THAT( arg, matcher ) (void)(0)
17898 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17899
17900 #define INFO( msg ) (void)(0)
17901 #define UNSCOPED_INFO( msg ) (void)(0)
17902 #define WARN( msg ) (void)(0)
17903 #define CAPTURE( ... ) (void)(0)
17904
17905 #define TEST_CASE( ... ) INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_S_T_ ))
17906 #define TEST_CASE_METHOD( className, ... )
INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ))
17907 #define METHOD_AS_TEST_CASE( method, ... )
17908 #define REGISTER_TEST_CASE( Function, ... ) (void)(0)
17909 #define SECTION( ... )
17910 #define DYNAMIC_SECTION( ... )
17911 #define FAIL( ... ) (void)(0)
17912 #define FAIL_CHECK( ... ) (void)(0)
17913 #define SUCCEED( ... ) (void)(0)
17914 #define ANON_TEST_CASE() INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_S_T_ ))
17915
17916 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
17917 #define TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION(__VA_ARGS__)
17918 #define TEMPLATE_TEST_CASE_SIG( ... )
INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION(__VA_ARGS__)
17919 #define TEMPLATE_TEST_CASE_METHOD( className, ... )
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION(className, __VA_ARGS__)

```

```

17920 #define TEMPLATE_TEST_CASE_METHOD_SIG( className, ... )
        INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION(className, __VA_ARGS__ )
17921 #define TEMPLATE_PRODUCT_TEST_CASE( ... ) TEMPLATE_TEST_CASE( __VA_ARGS__ )
17922 #define TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) TEMPLATE_TEST_CASE( __VA_ARGS__ )
17923 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) TEMPLATE_TEST_CASE_METHOD( className,
        __VA_ARGS__ )
17924 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) TEMPLATE_TEST_CASE_METHOD( className,
        __VA_ARGS__ )
17925 #else
17926 #define TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS(
        INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION(__VA_ARGS__) )
17927 #define TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS(
        INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION(__VA_ARGS__) )
17928 #define TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
        INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION(className, __VA_ARGS__ ) )
17929 #define TEMPLATE_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
        INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION(className, __VA_ARGS__ ) )
17930 #define TEMPLATE_PRODUCT_TEST_CASE( ... ) TEMPLATE_TEST_CASE( __VA_ARGS__ )
17931 #define TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) TEMPLATE_TEST_CASE( __VA_ARGS__ )
17932 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) TEMPLATE_TEST_CASE_METHOD( className,
        __VA_ARGS__ )
17933 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) TEMPLATE_TEST_CASE_METHOD( className,
        __VA_ARGS__ )
17934 #endif
17935
17936 #define STATIC_REQUIRE( ... ) (void)(0)
17937 #define STATIC_REQUIRE_FALSE( ... ) (void)(0)
17938
17939 #endif
17940
17941 #define CATCH_TRANSLATE_EXCEPTION( signature ) INTERNAL_CATCH_TRANSLATE_EXCEPTION_NO_REG(
        INTERNAL_CATCH_UNIQUE_NAME( catch_internal_ExceptionTranslator ), signature )
17942
17943 // "BDD-style" convenience wrappers
17944 #define SCENARIO( ... ) INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
        C_A_T_C_H_T_E_S_T_ ) )
17945 #define SCENARIO_METHOD( className, ... )
        INTERNAL_CATCH_TESTCASE_METHOD_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ),
        className )
17946
17947 #define GIVEN( desc )
17948 #define AND_GIVEN( desc )
17949 #define WHEN( desc )
17950 #define AND_WHEN( desc )
17951 #define THEN( desc )
17952 #define AND_THEN( desc )
17953
17954 using Catch::Detail::Approx;
17955
17956 #endif
17957
17958 #endif // ! CATCH_CONFIG_IMPL_ONLY
17959
17960 // start catch_reenable_warnings.h
17961
17962
17963 #ifdef __clang__
17964 #   ifdef __ICC // icpc defines the __clang__ macro
17965 #       pragma warning(pop)
17966 #   else
17967 #       pragma clang diagnostic pop
17968 #   endif
17969 #elif defined __GNUC__
17970 #   pragma GCC diagnostic pop
17971 #endif
17972
17973 // end catch_reenable_warnings.h
17974 // end catch.hpp
17975 #endif // TWOBLUECUBES_SINGLE_INCLUDE_CATCH_HPP_INCLUDED

```

## 5.2 /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/cmdline.h

```

00001 //
00002 // Created by Lindsay Haslam on 1/11/24.
00003 //
00004
00005 #ifndef EXPRESSIONCLASSES_CMDLINE_H
00006 #define EXPRESSIONCLASSES_CMDLINE_H
00007
00008 #include "catch.h"
00009 #include <cstring>
00010 #include <iostream>
00011

```

```
00012 //class cmdline: public main {
00013 //
00014 //};
00015
00016 void use_arguments(int argc, char **argv);
00017
00018
00019 #endif //HW1_CMDLINE_H
```

## 5.3 /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.cpp File Reference

This header file declares the [Expr](#) class hierarchy for representing arithmetic expressions. It includes the abstract base class [Expr](#) and its derived classes [Num](#), [Var](#), [Add](#), and [Mult](#). Each class implements functionalities for equality comparison, interpretation (evaluation), variable presence checking, substitution, pretty printing, and standard printing of expressions. This architecture allows for the representation and manipulation of complex arithmetic expressions involving numbers, variables, and the operations of addition and multiplication.

```
#include "Expr.h"
```

### 5.3.1 Detailed Description

This header file declares the [Expr](#) class hierarchy for representing arithmetic expressions. It includes the abstract base class [Expr](#) and its derived classes [Num](#), [Var](#), [Add](#), and [Mult](#). Each class implements functionalities for equality comparison, interpretation (evaluation), variable presence checking, substitution, pretty printing, and standard printing of expressions. This architecture allows for the representation and manipulation of complex arithmetic expressions involving numbers, variables, and the operations of addition and multiplication.

#### Author

Lindsay Haslam

#### Date

1/18/24

## 5.4 /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.h File Reference

This header file defines a hierarchy of expression classes for representing and manipulating arithmetic expressions.

```
#include <stdlib.h>
#include <stdio.h>
#include <string>
#include <stdexcept>
#include <sstream>
```

## Classes

- class [Expr](#)
- class [Num](#)
- class [Var](#)
- class [Add](#)
- class [Mult](#)

## Enumerations

- enum `precedence_t` { `prec_none` , `prec_add` , `prec_mult` }

### 5.4.1 Detailed Description

This header file defines a hierarchy of expression classes for representing and manipulating arithmetic expressions.

#### Author

Lindsay Haslam

#### Date

1/18/24

The [Expr](#) class hierarchy includes classes for numerical values ([Num](#)), variables ([Var](#)), addition operations ([Add](#)), and multiplication operations ([Mult](#)). Each class provides methods for equality checks, interpretation (evaluation), variable presence checks, substitution, and printing. The design supports the construction and manipulation of arithmetic expressions involving integers, variables, addition, and multiplication, allowing for pretty-printing and evaluation with variable substitution.

## 5.5 /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.h

[Go to the documentation of this file.](#)

```

00001
00009 #ifndef EXPRESSIONCLASSES_EXPR_H
00010 #define EXPRESSIONCLASSES_EXPR_H
00011
00012 #include <stdlib.h>
00013 #include <stdio.h>
00014 #include <string>
00015 #include <stdexcept>
00016 #include <sstream>
00017
00018 using namespace std;
00019
00020 typedef enum {
00021     prec_none,      // = 0
00022     prec_add,       // = 1
00023     prec_mult       // = 2
00024 } precedence_t;
00025
00026 class Expr {
00027 public:
00028     virtual bool equals(Expr *e) = 0;
00029     virtual int interp() = 0;
00030     virtual bool has_variable() = 0;
00031     virtual Expr* subst(string varName, Expr* replacement) = 0;
00032     virtual void print (ostream& os) = 0;
00033
00034     string to_string();

```

```

00035     void pretty_print(ostream &ostream);
00036     virtual void pretty_print_at(ostream &ot, precedence_t prec);
00037     string to_pretty_string();
00038 };
00039
00040 class Num : public Expr {
00041 public:
00042     int val;
00043     Num(int val);
00044     bool equals(Expr *e);
00045     //Return the value
00046     int interp();
00047     //Num will never have a variable.
00048     bool has_variable();
00049     Expr* subst( string varName, Expr* replacement);
00050     virtual void print (ostream& os);
00051     string to_string();
00052 };
00053
00054 class Var : public Expr{
00055 public:
00056     string name;
00057     Var(string name);
00058     bool equals(Expr *e);
00059     int interp();
00060     //Will have a variable.
00061     bool has_variable();
00062     Expr* subst( string varName, Expr* replacement);
00063     virtual void print (ostream& os);
00064 };
00065
00066 class Add : public Expr {
00067 public:
00068     Expr *lhs, *rhs;
00069     Add(Expr *lhs, Expr *rhs);
00070     bool equals(Expr *e);
00071     //Sum of the subexpression values
00072     int interp();
00073     //Check if either have a variable
00074     bool has_variable();
00075     Expr* subst( string varName, Expr* replacement);
00076     virtual void print (ostream& os);
00077     void pretty_print_at(ostream &ot, precedence_t prec);
00078 };
00079
00080 class Mult : public Expr {
00081 public:
00082     Expr *lhs, *rhs;
00083     Mult(Expr *lhs, Expr *rhs);
00084     bool equals(Expr *e);
00085     //The product of the subexpression values
00086     int interp();
00087     //Check if either have a variable
00088     bool has_variable();
00089     Expr* subst(string varName, Expr* replacement);
00090     virtual void print (ostream& os);
00091     void pretty_print_at(ostream &ot, precedence_t prec);
00092 };
00093
00094 #endif //EXPRESSIONCLASSES_EXPR_H
00095

```

## 5.6 /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr Tests.cpp File Reference

This test file contains a series of test cases for the `Expr` class hierarchy, focusing on `Var`, `Num`, `Add`, and `Mult` classes.

```
#include "ExprTests.h"
```

### Functions

- `TEST_CASE` ("Expr Var")



- **TEST\_CASE** ("Var Interp Throws")
- **TEST\_CASE** ("Var Has Variable")
- **TEST\_CASE** ("Var Subst")
- **TEST\_CASE** ("Expr Add")
- **TEST\_CASE** ("Add Interp")
- **TEST\_CASE** ("Add Has Variable")
- **TEST\_CASE** ("Add Subst")
- **TEST\_CASE** ("Expr Num")
- **TEST\_CASE** ("Num Interp")
- **TEST\_CASE** ("Num Has Variable")
- **TEST\_CASE** ("Expr Mult")
- **TEST\_CASE** ("Mult Interp")
- **TEST\_CASE** ("Mult Has Variable")
- **TEST\_CASE** ("Mult Subst")
- **TEST\_CASE** ("Nabil's Tests for Pretty Print")
- **TEST\_CASE** ("Pretty Print [Mult](#) Expressions")
- **TEST\_CASE** ("Pretty Print [Add](#) Expressions")
- **TEST\_CASE** ("Pretty Print [Var](#) Expressions")
- **TEST\_CASE** ("Multiple Types of Operations")

### 5.6.1 Detailed Description

This test file contains a series of test cases for the [Expr](#) class hierarchy, focusing on [Var](#), [Num](#), [Add](#), and [Mult](#) classes.

#### Author

Lindsay Haslam

#### Date

1/18/24

It includes tests for equality checks, interpretation (evaluation), variable presence checks, substitution, and pretty printing functionalities of arithmetic expressions. Each test case is designed to verify the correct behavior of the classes and their interactions, ensuring that expressions are correctly manipulated and evaluated according to the rules of arithmetic and variable substitution.

## 5.7 /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/ExprTests.h

```
00001 //
00002 // Created by Lindsay Haslam on 1/18/24.
00003 //
00004
00005 #ifndef EXPRESSIONCLASSES_EXPRTESTS_H
00006 #define EXPRESSIONCLASSES_EXPRTESTS_H
00007
00008 #include "catch.h"
00009 #include "Expr.h"
00010
00011
00012 #endif //EXPRESSIONCLASSES_EXPRTESTS_H
```

## 5.8 /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/main.cpp File Reference

Main entry point for the program. This file contains the main function that serves as the entry point of the program.

```
#include "cmdline.h"
```

### Functions

- int **main** (int argc, char \*\*argv)

### 5.8.1 Detailed Description

Main entry point for the program. This file contains the main function that serves as the entry point of the program.

#### Author

Lindsay Haslam

#### Date

2/6/2024